FES – Fractal Encryption Standard

Peer Review Guide

FES Framework

February 27, 2026

Contents

1 Purpose of This Document **3**

2 Three-Stage Architecture **3**
2.1 Stage 1: Password to Fractal Portal . . . 3
2.2 Stage 2: Fractal Portal to Transform Stream . . . 3
2.2.1 Optional: Reorder Stream (Scramble) . . . 4
2.3 Stage 3: Overwrite and Payload Transformation . . . 4
2.3.1 Multi-Pass Amplification . . . 10

3 System Overview **10**
3.1 High-Level Transform . . . 10

4 Password to Portal Mapping **11**

5 Fractal Stream Generator **11**
5.1 Geometric Navigation . . . 11

6 Stream Extraction **12**

7 Scramble Mode **12**
7.1 Sort Array Construction . . . 12
7.2 Permutation Derivation . . . 12
7.3 Tie Handling . . . 12
7.4 Payload Reordering . . . 12
7.4.1 Illustrative Excerpt: Per-Pass Sort Array . . . 13

8 Silo Architecture **13**
8.1 Purpose of Silos . . . 13
8.2 Portal Selection Within a Silo . . . 13
8.3 Dimensional Expansion . . . 14

8.4 Master and Organization Silos 14
8.5 Default Silo Mode 14
8.6 Security Implications 14

9 Security Objectives **15**

10 Threat Model **15**

11 Definition of a Break **16**

12 Conclusion **16**

1 Purpose of This Document

This document defines the formal structure and evaluation framework for independent peer review of the Fractal Encryption System (FES).

FES is architecturally distinct from classical block cipher paradigms. It is a deterministic, password-parameterized, fractal-field navigation system that generates transformation streams via multi-dimensional Mandelbrot dynamics.

This document provides sufficient structural clarity for cryptographic, mathematical, and computational scrutiny.

2 Three-Stage Architecture

FES operates in three distinct stages:

1. Password to Fractal Portal
2. Fractal Portal to Transform Stream (and optional Reorder Streams)
3. Payload Transformation via Overwrite Operators

2.1 Stage 1: Password to Fractal Portal

Password material, combined with configuration parameters (Silo, FOTP, dimensionality, etc.), is deterministically mapped to a Fractal Portal.

$$Z_0 = \Phi(\text{Password}, \text{Config}, \text{Silo}, \text{FOTP})$$

The mapping performs:

- Base vector selection from Silo set (4 hex bytes per dimension pair)
- Offset derivation for x and y pairs in multiple dimensions (14 hex bytes each)
- Addition of offsets to selected Silo vectors

After Z_0 is constructed, raw password material is discarded and is not applied to the payload.

2.2 Stage 2: Fractal Portal to Transform Stream

From Z_0, the multi-dimensional Mandelbrot function evolves:

$$Z_{k+1} = MB_D(Z_k)$$

Each iteration produces an array of z values.

Each z is locally consumed to derive:

- Angle θ
- Hypotenuse r

Using Pythagorean navigation:

$$x_i' = x_i + r_i \cos(\theta_i)$$
$$y_i' = y_i + r_i \sin(\theta_i)$$

Decimal z outputs are converted into byte sequences forming the Transform Stream:

$$K^{(t)} = [k_0^{(t)}, \ldots, k_{n-1}^{(t)}]$$

for pass t.

2.2.1 Optional: Reorder Stream (Scramble)

If Scramble is enabled:

- Raw decimal z values are collected into a per-pass array
- Array size equals payload length
- Sorting produces permutation $\pi^{(t)}$
- Ties resolved by sequence order

Payload is reordered per pass:

$$P_{\text{scr}}^{(t)}[j] = P^{(t)}[\pi^{(t)}(j)]$$

Each pass generates a new permutation.

2.3 Stage 3: Overwrite and Payload Transformation

The Transform Stream $K^{(t)}$ is applied to the payload using a configurable operator set O.

Operators are applied in fixed order per pass.
Available operators include:

xor Bitwise XOR between payload byte and stream byte. Default operator.

add Modulo addition of payload byte and stream byte.

sub – Fractal-Driven Byte Substitution The substitution operator performs a deterministic but dynamically fractal-conditioned byte substitution.

Step 1: Seed Initialization A base array B of length 256 is initialized:

$$B = [0, 1, 2, \ldots, 255]$$

Step 2: Deterministic Mixing Using a fixed starting location, 2550 pairwise swaps are performed. Swap positions are selected using a deterministic pseudo-random generator.

This produces a fixed mixed permutation:

$$B_{\text{seed}}$$

This seed permutation is constant across executions.

Step 3: Working Array Construction For each pass, the seed permutation is copied:

$$B_{\text{work}} = B_{\text{seed}}$$

Step 4: Pass-Specific Permutation via Sort Array. If `sub` is enabled, FES derives a per-pass permutation over $\{0, \ldots, 255\}$ using the same stable-rank mechanism as Scramble.

For each pass t, a 256-length order vector is constructed:

$$S_{\text{sub}}^{(t)} \in \mathbb{R}^{256}$$

and converted to a permutation $\pi_{\text{sub}}^{(t)}$ by sorting with stable tie handling (original sequence order).

A per-pass substitution table is then defined by permuting a seed deck:

$$B_{\text{work}}^{(t)}[j] = B_{\text{seed}}[\pi_{\text{sub}}^{(t)}(j)]$$

Short Payload Handling. If payload length $n < 256$, FES still generates $S_{\text{sub}}^{(t)}$ of length 256 when `sub` is enabled, ensuring substitution table construction is independent of payload size.

Step 5: Encryption Substitution For payload byte $p(n)$ and fractal stream byte $k(n)$:

$$i = (p(n) + k(n)) \bmod 256$$

$$p'(n) = B_{\text{work}}[i]$$

Thus substitution is dependent on:

- The fixed seed permutation
- Fractal-stream-driven restructuring
- Per-byte fractal stream value

The substitution mapping changes per pass due to fractal-driven restructuring of B_{work}.

fBlit – Prime Fractal Bit Swaps with Scaled Offset The `fBlit` overwrite operator performs stream-driven swaps of prime-length bit blocks within the payload.

Let the payload bit-length be:

$$L = 8 \cdot n$$

where n is payload length in bytes. The operator exits if $n < 9$ (i.e., if UBound(payload) $<$ 8).

Prime Parameters Two thresholds are computed:

$$p_{\text{split}} = \left\lfloor \frac{L}{4} \right\rfloor - 1, \qquad p_{\max} = \left\lfloor \frac{L}{5} \right\rfloor$$

A prime list $\mathcal{P}$ is provided (e.g., precomputed primes). A *fixed prime* p_f is selected as the largest prime below susceptibility threshold p_{split} (implementation selects the previous prime when crossing the threshold). If thresholds exceed available primes, a fallback selection is derived from stream material.

A *maximum prime index* $I_{\max}$ is chosen as the largest index such that $\mathcal{P}[I_{\max}] < p_{\max}$ (with an implementation fallback if not found).

Static ZAP Phase (Two Deterministic Swaps) On encryption, two deterministic swaps are performed using the fixed prime p_f:

$$\text{SwapBits}(P, 0, p_f, p_f)$$

$$\text{SwapBits}(P, L - 2p_f, L - p_f, p_f)$$

On decryption, these two swaps are applied in reverse order at the end.

Dragon Phase (Stream-Driven Prime Swaps) Let the number of swaps be bounded by:

$$N = \min\left(256, \left\lfloor \frac{L}{100} \right\rfloor\right)$$

For each swap $j \in \{0, \ldots, N-1\}$, four stream bytes are consumed (at indices $4j, 4j+1, 4j+2, 4j+3$ for the given pass).

A prime index is selected:

$$i_p = k_{4j} \bmod (I_{\max} + 1) \qquad \Rightarrow \qquad p = \mathcal{P}[i_p]$$

A source position (bit offset) is selected and scaled to payload length:

$$\text{src} = \left\lfloor \frac{k_{4j+1} \cdot (L - p)}{128} \right\rfloor$$

with boundary correction to ensure:

$$0 \leq \text{src} \leq L - p - 1$$

A direction is chosen based on which half of the payload src falls into:

$$d = \begin{cases} +1 & \text{if src} < \frac{L}{2} \\ -1 & \text{otherwise} \end{cases}$$

An offset scaled by the prime length is derived:

$$\Delta = \left\lfloor \frac{k_{4j+3} \cdot p}{128} \right\rfloor$$

and the target position is computed:

$$\text{tgt} = \text{src} + d \cdot (p + \Delta)$$

with boundary correction to ensure:

$$0 \leq \text{tgt} \leq L - p - 1$$

Finally, a prime-length bit-block swap is performed:

$$\text{SwapBits}(P, \text{src}, \text{tgt}, p)$$

Invertibility The operator is invertible.

Encryption applies swaps for $j = 0$ to $N - 1$. Decryption replays the same swap schedule in reverse order ($j = N - 1$ down to 0), followed by reversing the static ZAP phase.

bit split – Fractal Stream Driven Bit-Split Rotation (Byte-Local) The `bit split` overwrite operator performs a per-byte bit rearrangement driven by the fractal transform stream.

For payload byte $p(n) \in \{0, \ldots, 255\}$ and stream byte $k(n)$, a split point s is derived:

$$s = k(n) \bmod 7$$

If $s = 0$, no change is applied.
Otherwise, the byte is partitioned into:

- a lower bit-field of width $(8 - s)$
- an upper bit-field of width s

Define:

$$\text{mask}_{\text{low}} = 2^{(8-s)} - 1$$

$$\text{low} = p(n) \;\&\; \text{mask}_{\text{low}}$$

$$\text{high} = \left\lfloor \frac{p(n) \;\&\; (255 - \text{mask}_{\text{low}})}{2^{(8-s)}} \right\rfloor$$

The output byte is formed by swapping these fields:

$$p'(n) = (\text{low} \cdot 2^s) \;\; \text{OR} \;\; \text{high}$$

This operation is equivalent to a stream-conditioned rotation of the byte by s bits (under the implemented field widths), performed independently for each byte position.

Notes. The implementation uses $s = k(n) \bmod 7$ (rather than mod8) to avoid extreme split points and the trivial full-byte rotation case.

bit warp – Prime-Length Bit-Block Rotation The `bit warp` overwrite operator performs stream-driven bit-level rearrangement of the payload by iteratively selecting prime-length bit blocks and rotating their internal partition.

Let the payload bit-length be:

$$L = 8 \cdot n$$

where n is the payload length in bytes.
If $L < 64$, the operator exits without modification.

Stream-Driven Prime Selection For each step, two prime values are derived from the transform stream:

- p_ℓ: a *prime chunk length* selected from a prime set (e.g., primes in a configured range such as 19–31)
- s_p: a *split point* selected from a prime set (e.g., primes in a configured range such as 3–29)

These are obtained via a function of the stream and a stream index, advancing deterministically through stream material.

In reverse mode (for decryption), the split point is inverted:

$$s_p \leftarrow p_\ell - s_p$$

Sequential Bit-Block Processing Starting from bit position $\text{pos} = 0$, the operator processes bit blocks sequentially:

$$\text{pos} \leftarrow \text{pos} + p_\ell$$

until insufficient bits remain for another block.

If $\text{pos} + p_\ell > L$, processing of full blocks terminates and a final remainder block may be processed.

Bit-Block Extraction and Rotation Let:

$$B = \text{GetBitBlock}(P, \text{pos}, p_\ell)$$

be the integer value represented by the p_ℓ bits starting at bit offset pos.

Define a partition of B at split point s_p:

$$B = L_{\text{part}} \cdot 2^{p_\ell - s_p} + R_{\text{part}}$$

where:

$$L_{\text{part}} = \left\lfloor \frac{B}{2^{p_\ell - s_p}} \right\rfloor, \quad R_{\text{part}} = B \bmod 2^{p_\ell - s_p}$$

The operator then swaps (rotates) these partitions:

$$B' = R_{\text{part}} \cdot 2^{s_p} + L_{\text{part}}$$

and writes back:

$$\text{SetBitBlock}(P, \text{pos}, p_\ell, B')$$

This is a deterministic, fractal-stream-conditioned bit permutation applied across the payload.

Tail Handling After full block processing, if a remainder block exists (e.g., $L - \text{pos} > 6$ bits), the remaining bits may be processed as a final block to avoid leaving a small tail untransformed.

Invertibility The operator is invertible.

In decryption, the same p_ℓ sequence is generated and the split point is inverted by:

$$s_p \leftarrow p_\ell - s_p$$

which reverses the partition rotation and restores the original payload bits.

2.3.1 Multi-Pass Amplification

For T passes:

$$P^{(t+1)} = O(P^{(t)}, K^{(t)})$$

Each pass:

- Generates fresh fractal transform stream
- Generates fresh sort/scramble permutation (if enabled)
- Reapplies selected overwrite operators

Thus amplification occurs across passes, each with unique fractal stream segments.

3 System Overview

3.1 High-Level Transform

Let:

$$P \in \{0,1\}^n$$

be a payload of length n bytes.
FES transforms P into ciphertext C via:

$$C = F(P; \text{Portal}, D, T, O)$$

where:

- Portal = fractal entry coordinates derived from password material
- D = dimensionality (minimum 8, configurable)
- T = number of passes
- O = selected operator set

4 Password to Portal Mapping

Password material is not expanded into persistent internal key state.

Instead:

Password → Fractal Portal Mapping → Password Discarded → Fractal Iteration → Fractal Stream

Let:

$$Z_0 = \Phi(\text{Password}, \text{Config}, \text{Silo}, \text{FOTP})$$

where Φ deterministically maps input material to initial multi-dimensional coordinates.

After Z_0 is constructed, raw password material Password, Config, Silo and FOTP is discarded. It is not retained and is not directly applied to payload bytes.

5 Fractal Stream Generator

FES employs a multi-dimensional Mandelbrot function:

$$Z_{k+1} = MB_D(Z_k)$$

where Z_k is an array of complex vectors across D dimensions.

5.1 Geometric Navigation

Each iteration produces an array of z values.

Each z is locally consumed to derive:

- Angle θ
- Hypotenuse r

For each dimension pair (x_i, y_i):

$$x_i' = x_i + r_i \cos(\theta_i)$$

$$y_i' = y_i + r_i \sin(\theta_i)$$

Thus:

$$Z_k \to \{r_i, \theta_i\}_{i=1}^{D/2} \to Z_{k+1}$$

Each z value is consumed locally and not reused globally.

6 Stream Extraction

Decimal values produced during navigation are transformed into byte sequences.

In current development versions, 12 significant bytes are extracted per iteration per dimension via a memory transfer.

Stream length per pass is:

$$|K^{(t)}| = n$$

Total stream length across passes:

$$n \times T$$

7 Scramble Mode

If Scramble is enabled, a per-pass reordering is applied.

7.1 Sort Array Construction

During pass t, raw decimal z values are collected into:

$$S^{(t)} = [s_0^{(t)}, s_1^{(t)}, \ldots, s_{n-1}^{(t)}]$$

where n is payload length.

Each pass produces a new array $S^{(t)}$.

7.2 Permutation Derivation

A permutation $\pi^{(t)}$ is derived by sorting $S^{(t)}$:

$$\pi^{(t)} = \text{rank}(S^{(t)})$$

7.3 Tie Handling

If equal values occur under finite decimal representation, ties are resolved deterministically using original sequence order.

This guarantees $\pi^{(t)}$ is a bijection.

7.4 Payload Reordering

Payload bytes are reordered:

$$P_{\text{scr}}^{(t)}[j] = P^{(t)}[\pi^{(t)}(j)]$$

Permutation differs per pass.

7.4.1 Illustrative Excerpt: Per-Pass Sort Array

Table 1 illustrates an excerpt of the per-pass decimal Sort Array values $s_i^{(t)}$ for payload indices $i = 0..31$ over two passes.

Byte #	Pass 1 $s_i^{(1)}$	Pass 2 $s_i^{(2)}$
0	4793.4512063949561677	3849.1662474244608959
1	357.7112770234736040	611.4949519204787601
2	1447.6222373512767626	3077.2004738578931151
3	822.5561887333744784	2817.9948358302237401
4	2585.3478635626196090	4279.9316279229324200
5	185.8660109157144254	235.4281698169864171
6	3626.0061026419645930	47.7679766295334759
7	205.1596320617041240	31.6675759128826838
8	37.3888294878806581	2.5589124980107219
⋮	⋮	⋮
31	656.2230266658864937	15369.3432006339994286

Table 1: Excerpt of Scramble Sort Array values by byte index for two passes.

This is also used for substitution (sub) lookup per pass permutation reordering.

8 Silo Architecture

8.1 Purpose of Silos

A Silo defines a deployment-specific fractal entry domain.

Silos prevent cross-deployment transferability of attack knowledge. An attack model constructed against one Silo does not apply to another.

Formally, a Silo defines a bounded set of initial coordinate vectors:

$$\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{2^{16}}$$

Each Silo contains 2^{16} (x, y) vector pairs. These vectors are derived using a GUID-based generation process, ensuring practical uniqueness across Silos.

8.2 Portal Selection Within a Silo

Password material does not generate coordinates directly.

Instead, password material selects:

- A base vector from $\mathcal{S}$ using 4 bytes of hex per dimension pair
- An x offset derived from 14 hex bytes
- A y offset derived from 14 hex bytes

These offsets are added to the selected Silo vector:

$$(x', y') = (x_{\text{Silo}} + \Delta x,\ y_{\text{Silo}} + \Delta y)$$

This produces the Fractal Portal coordinate.

8.3 Dimensional Expansion

For dimensionality D:

- Each dimension pair is independently selected from the Silo set.
- Minimum $D = 8$.
- Each dimension pair contributes 224 bits of coordinate precision.

Total state width scales linearly with dimensionality. The number of dimensions is configurable and unbounded. FES enforces a minimum of 8 dimensions.

8.4 Master and Organization Silos

FES supports hierarchical Silo models:

- Master Silo
- Organization Silos encrypted under the Master Silo

Organization Silos are decrypted in memory only and stored as decimal arrays during runtime.

They are never written to disk in decrypted form.

8.5 Default Silo Mode

FES includes a default Silo. If no Silo is configured, the system operates using the built-in default.

Security-critical deployments should not rely on the default Silo.

8.6 Security Implications

Silos provide:

- Deployment Isolation
- Cryptographic Compartmentalization
- Fractal entry domain separation
- Cross-domain non-transferability of attacks

- Expanded search space when Silo is unknown

If a Silo is unknown to an adversary, brute-force attempts cannot be meaningfully executed. Each Silo defines a distinct coordinate domain and portal root within the multi-dimensional fractal field. As a result, identical passwords under different Silos generate entirely different fractal streams and transformation behaviour. An incorrect Silo selection yields a structurally unrelated transform stream, preventing reliable verification of password guesses.

If a Silo is compromised, security reduces to password, FOTP, configuration, and dimensional parameters.

9 Security Objectives

Peer review should evaluate:

1. Keystream unpredictability
2. Non-transferability of known plaintext
3. Whole-of-payload encryption coupling
4. Oracle suppression
5. Entropy preservation
6. Silo deployment isolation

10 Threat Model

Analysis should consider:

- Known plaintext
- Chosen plaintext
- Chosen ciphertext
- Multi-message reuse
- Configuration disclosure
- Silo compromise
- Neural approximation attacks
- Cycle detection
- State reconstruction

11 Definition of a Break

FES is considered broken if:

- Key recovery faster than brute force
- Practical distinguisher
- State reconstruction
- Configuration inference
- Effective correctness oracle
- Transferable cross-Silo attack

12 Conclusion

FES proposes a deterministic fractal-field navigation model for whole-of-payload encryption, distinct from classical block cipher architectures.

Security claims must be evaluated through formal attack modeling, empirical analysis, and adversarial testing.

This document defines the structure necessary for objective peer review.

Resources

For additional information, tools, and testing interfaces related to the Fractal Encryption System (FES), the following resources are available:

- FES Framework — official overview, download, and documentation
- Portalz Demo — interactive online demo of FES encryption
- `peer-review@portalz.solutions` — dedicated contact for peer review inquiries