A JOURNEY OF
CURIOSITY.
A PATH OF
DISCOVERY.
A NEW STANDARD
EMERGED.

THE FES JOURNEY

THE ENGINEERING JOURNEY THAT LED TO THE
FRACTAL ENCRYPTION STANDARD (FES)

A COMPANION PAPER TO
New Cryptographic Art & Science
portalz.solutions/FES-CAS.html

OBSERVE. · QUESTION. · DISCOVER. · INNOVATE. · SECURE.

Contents

Chapter One: A Serendipitous Journey 3
The Twist 3
I Found the Galaxonomy ID 3
Fractal Space Discoveries 3
Twelve Months of Nothing Satisfying 4
Yamba 4
Standing It Down 4
An Overheard Claim 5
The Year I Stopped Everything 5
An Elephant, Sighted 6
Chapter Two: The Word Nobody Is Allowed to Use 7
The Bump 7
Finding Footing Inside the Curriculum 8
Chapter Three: Systems Analysis 9
A Ceiling I Could See 9
Performance 10
Core Overwrite 11
Secondary Overwrite 12
Chapter Four: Silos 12
Genesis 12
From Infinity Exclusion to Silo 13
Silos Solve Encryption Monoculture 13
Private Cryptographic Foundations 14
Chapter Five: Fractal OTP 14
Meeting Shannon's OTP Requirements 14
Verifying FES OTP Requirements 15
Chapter Six: Overwrite Passes 16
Fractal Stream Continuation 16
Contrast to AES multi-pass 17
Overwrite Options Order 17
Chapter Seven: FES Matured 18
New Cryptographic Art & Science 18

Copyright © 2026 Wolfgang Flatow

The FES Journey

Chapter One: A Serendipitous Journey

A serendipitous journey pursued with bloody-minded persistence, based on professional curiosity. That's the whole of it, if I had to say it in one line. Everything else here is just the proof.

I want to say this plainly, before anything else: I never set out to compete with AES. I never went looking for a way to beat it, embarrass it, or replace it. AES was the king. I'd use it myself, without a second thought. If this story reads at any point like a challenge to that standing, I've told it wrong.

What actually happened is different than that, and stranger.

The Twist

It started nowhere near cryptography. I am a systems analyst and enterprise architect, working on Galaxonomy — a universal taxonomy problem, the kind of thing that sounds dry until you're the one staring at it. Taxonomy identifiers have to do something particular: they can't collide, and they have to scale without the whole structure buckling as it grows. I was deep in that problem, thinking the way that kind of problem makes you think, when something in it twisted sideways. What had looked like a naming problem started looking like an ID problem. And somewhere in that shift, fractals entered a conversation they had no obvious business being in.

I didn't go looking for fractals. They arrived because I was looking hard enough at something else.

I Found the Galaxonomy ID

The fractal R&D yielded a method of constructing IDs using x,y regions to contain categories. That was the actual answer to the problem I'd started with — a way to build identifiers that didn't collide and didn't buckle as the taxonomy grew, using fractal regions as containers for categories rather than the flat numbering schemes I'd been fighting against. Galaxonomy had its ID.

It should have ended there. The problem I'd set out to solve was solved.

Fractal Space Discoveries

But something in the fractal math itself kept pulling at my attention, past the point where the ID problem needed it to. Fractals aren't just good at rendering striking images — sitting inside

Copyright © 2026 Wolfgang Flatow

that math is a deep, genuine complexity: unpredictable, effectively infinite in its detail no matter how far you zoom. I noticed that, and it didn't fit neatly back into the taxonomy box I'd found it useful for. It looked like it might be useful for something else entirely — encryption — precisely because of that unpredictability and that infinite complexity.

So I started investigating how that might actually be leveraged. Not because Galaxonomy needed it. Because the question itself wouldn't let go of me.

Twelve Months of Nothing Satisfying

For about a year after that, I carried the question around without an answer. Elements of it surfaced now and then — a property here, a possibility there — but nothing assembled. Nothing cohered into a system I could point to and say: that works. It wasn't urgent. Nobody needed me to solve it. I kept at it anyway, the way you keep turning a stone over because the shape of it bothers you, not because anything depends on it. Pure R&D bloody-mindedness. This looks different — is there something here?

Yamba

Then it arrived, essentially whole, and I know exactly where I was when it did — across the river from where I happened to be holidaying as I write this, in fact. Three stages, and no fewer:

1. Fractal Portal Mapping — the password or key material gets used once, then discarded. It doesn't survive as itself.
2. Fractal Stream Generation — a stream built to match the payload's own length, however many passes that takes.
3. Payload Overwrite — the stream applied to the payload itself, whole, not in blocks.

Omit any one of the three and there is no system. That's not a design preference; it's the nature of the thing. I coded it, tested it, and it worked, the same day the shape of it became clear. I called the prototype Yamba, after the town.

Only later did I understand what the third stage quietly implied. A fractal stream matched to the full payload length means the payload is never addressed a block at a time — it never was, from the very first conception of the transformation. That wasn't a decision I made against block ciphers. It was just what a fractal stream is. The door it opened — that byte order was no longer a fixed structural fact but something I could choose — stood there unremarked for a long time before I thought to walk through it.

Standing It Down

And then I stopped.

Copyright © 2026 Wolfgang Flatow

Yamba worked. It was interesting. And I looked at it clearly and thought: AES is king, who am I to muck? There was no problem to solve, so there was nothing to keep pushing against. I set it down, the same plain instinct that had kept me at it for a year now telling me it was fine to let it rest.

I think that stand-down matters as much as anything I built. It's the proof that none of this was ever about proving AES was flawed. If it had been, I wouldn't have been able to walk away from a working result with no urge to keep going.

An Overheard Claim

2022 brought it back, and not through any plan of mine. Quantum computing videos were everywhere that year, and the claims in them made no distinction between kinds of encryption — they simply said quantum breaks encryption, full stop. I did not realize then that NIST did not include AES in its Post Quantum Encryption (PQE) program.

As far as I could analyze, quantum computers could access all combinations at once by superposition, an amazing and unique capability.

I just thought: AES would be vulnerable to quantum computers as they bypassed computational difficulty — so how would Yamba fare? A plain question, aimed at something I'd shelved a year earlier, prompted by a claim I had no way yet of knowing did not include AES.

I analyzed Yamba against quantum superposition. Yamba held up because the password is discarded at portal discovery and the fractal stream overwrites the whole payload. Quantum-proof, impenetrable — the word entered my vocabulary right there, years before it became the title of anything else.

It turned out NIST uses Grover’s threat exclusively to evaluate the quantum threat to AES, which places it decades in the future. In contrast, I was always looking at a completely different threat model – a qubit key-space in superposition.

The Year I Stopped Everything

That single test didn't stay a single test. I stopped everything else and went into quantum computing for over a year, as wide as I could make the study — not because Yamba demanded it, but because as soon as I started digging I could see where this was going. Quantum computers behave nothing like classical ones, and the doors they open aren't incremental, they're categorical. I looked at the state of the art — clumsy, early, transistor-stage — and recognized the same shape I'd recognized once before, decades earlier, watching renewable energy in its own transistor stage. Something that would compress and mature inside a working lifetime, mine included.

Copyright © 2026 Wolfgang Flatow

Fractal Transformation (FT), the Yamba prototype, stayed the anchor and the catalyst through that year, even as the study ran far wider than FT itself. I came out the other side able to call myself, honestly, a Quantum Security Analyst — not a title I set out to earn, one that simply became true along the way.

Somewhere in that year, I wrote my first working code that entangled a handful of qubits and put them into superposition to mimic a key-space. That's the moment the whole question stopped being something I'd heard about and became something I'd built and run myself. The same texture as Yamba, one field over: not just an idea, but an idea confirmed by my own hands making it work.

An Elephant, Sighted

Around 23 and 24, I let myself explore a handful of potential quantum cracks — five of them, roughly — purely to understand how such a thing would even be assembled. Not to build one. I was wary of that direction from the first moment it opened, and I stayed wary. Understanding the shape of a threat and manifesting one are not the same act, and I never confused them. I am not a hacker.

But in among that exploring, something plain came into focus, and it didn't need any of the surrounding complexity to be true:

A classical bit accesses the 2^{256} possibilities in a key-space one at a time. 256 qubits, properly entangled, hold all possibilities at once.

That's not a simplification. That is what a quantum computer is.

And once you sit with that plainly enough, a further question is unavoidable: if the whole key-space of possibilities (combinations) is already present, simultaneously, why would anyone search it at all? The task isn't finding the key among many — it's recognizing the one that's already there. AES will produce exactly one correct, sensibly structured plaintext for exactly one key, out of everything sitting in superposition. So the only real work left is building something that notices where it is.

I call this the Quantum Elephant in the Room. Not because it's hidden — because it isn't. Everyone in the field already knows qubits hold a key-space simultaneously. Everyone knows AES is a bijection with one correct answer per key. Both facts have been sitting in plain view the entire time the field has been discussing search-based attacks (Grover's) and doubling key lengths against them. Nobody had put the two facts in the same room and asked the obvious next question.

I didn't need a breakthrough to see it. I needed to look directly at something everyone could already see, and take it seriously.

Copyright © 2026 Wolfgang Flatow

This is where my quantum analysis lands — not with a proof, not with a crack, but with a plain recognition sitting quietly in a room full of people looking somewhere else. Everything since has been verifying this premise.

Chapter Two: The Word Nobody Is Allowed to Use

Once the quantum R&D was properly underway, an opinion started forming in me, and it hardened the longer I sat with it: AES, and encryption schemes like it, were quantum vulnerable. Key-space in superposition was the core of it — the same plain fact I'd already been circling. If a quantum register can hold the whole key-space at once, then a scheme's safety can't rest on the size of that space the way it used to. That reference point didn't leave me after I'd seen it. It became the lens everything else got checked against.

Including my own work.

So I turned the question on FES itself: how does it stack up against the same standard I was now applying to everything else? I didn't take the answer on faith. I went back through it properly — the fractal stream quality, analyzed at length, and the fact that the key is discarded at the very first stage, before the payload is ever touched. That discarding matters more than it might look like on the surface: there's no fixed transformation sitting behind the ciphertext for a quantum register to hold in superposition and check against, because the thing that would define that fixed transformation is already gone by the time encryption happens. I went over this ground more than once, from more than one angle, because I wanted it to fail if it was going to fail.

It didn't. Every analysis I ran said the same thing: FES would survive where AES would not.

I sat with that for a while before I let myself say the word. Eventually I did. I deduced that FES was impenetrable.

The Bump

And that's where I ran straight into something outside the mathematics entirely — a wall built out of the culture of the field itself, not out of any flaw I could find in the work.

Cryptography has a long, well-earned reflex: anyone who claims their scheme is unbreakable is, by that claim alone, treated as either naive or dishonest. It's not an unreasonable reflex. History is full of ciphers that were "impenetrable" right up until they weren't, and the field has learned, at real cost, to be suspicious of the word on sight. Claim impenetrability and the door closes before anyone reads the reasoning behind it.

I didn't understand that reflex at the time. I had no cryptography heritage to have absorbed it from — no years inside the field learning, the way anyone raised in it does, exactly which words

Copyright © 2026 Wolfgang Flatow

get you shown the door before your reasoning is even heard. I understand it now, looking back. But in the moment, I just had a finding I'd tried hard to break and couldn't, and a word that seemed like the plain, accurate description of it.

That's the bump this chapter is really about — not a technical wall, a cultural one. The moment where a finding, arrived at carefully and tested against itself more than once, runs headlong into a curriculum that has already decided, before any specific case is heard, that the word itself is the tell of a fraud.

Finding Footing Inside the Curriculum

But the wall didn't hold, not because I pushed harder against it, but because three things turned up, one after another, from inside the very curriculum that had thrown it up.

The first was Shannon. I hadn't come to any of this through the curriculum — I'd come sideways, through Galaxonomy and fractals and my own testing. So Shannon's work on secrecy systems wasn't something I'd absorbed early and carried with me; I found it later, already there, waiting inside the field as part of its foundation. That mattered. It meant there was a rigorous, agreed standard for what "unbreakable" would even have to mean, set down by someone the field already trusted completely — not my word against the reflex, but the reflex's own reference point.

More importantly, Shannon proved that impenetrability *is possible*. The only reason his method has not been adopted is that in its raw form it is not practical, especially keys that match the size of the payload.

The second was that FES, when I checked it properly against that standard, met Shannon's requirements — not as an abstract ideal, but in a practical, working framework. That's a different claim than "I think this is impenetrable." It's "this satisfies the same conditions the field itself already accepts as sufficient for perfect secrecy." The word stopped being mine to defend and became a description of a fit against Shannon’s proof.

The third was the sharpest of the three: every possible bit combination in an FES ciphertext decrypts to something valid — including every sensible-looking plaintext among them. There's no single correct answer sitting uniquely inside the ciphertext waiting to be recognized, the way there is with AES. An attacker holding the whole key-space in superposition, which in the case of FES is the same size as the ciphertext, has nothing to key on, because there's no unique structured output to distinguish the right key from all the wrong ones. The very property that makes the Elephant dangerous for AES — the one correct, structured plaintext — simply isn't there to exploit in FES.

That's where the outsider's ignorance of the reflex stopped being a liability. I hadn't learned to be afraid of the word, so I kept testing past the point where the field's own instinct would have

Copyright © 2026 Wolfgang Flatow

told me to stop and soften the claim. And the curriculum, once I actually went and read it properly, backed the word rather than punishing it.

Chapter Three: Systems Analysis

A Ceiling I Could See

There was still a plain limitation sitting in the Yamba FT, and I could see it clearly enough that it never became a blind spot. The key-state ran on only two fractal vectors — x and y. That was the whole of it, at the Yamba prototype. Each vector carried 112 bits, which put a hard ceiling on the key-space: 224 bits, total, no matter how the rest of the scheme performed.

224 bits is not a small number. But AES sat at 256, and I knew, looking at that gap plainly, that 224 would never do. Not because the difference was enormous — it isn't — but because there was no good reason to leave a ceiling in place once I could see it. If two vectors gave 224 bits, more vectors would give more. The fix wasn't subtle or clever. It was just obvious, once the limitation was named instead of politely ignored.

That's the seed of what came later — the move from two fractal dimensions to configurable, unlimited ones. But the naming of the ceiling belongs here, before the fix, because it's worth being honest that FES didn't start unbounded. It started at 224 bits, and the unlimited key-space was a deliberate answer to a limit I'd already spotted, not a feature that arrived by design from day one.

The answer wasn't a fix in the ordinary sense — it was a genuine insight, and it pointed away from the obvious route. Faced with a 224-bit ceiling, the obvious move would have been to push for more precision in x and y themselves — carry each vector out further, squeeze more bits per vector. That's not the direction I took. The insight was to add dimensions instead of accuracy: not two vectors carried to greater precision, but more vectors, each still carrying its own 112 bits. FES integrates an n-dimensional Mandelbrot, not a two-dimensional one, with no fixed number of dimensions built into the scheme. Two vectors had given 224 bits because two was all there was. Once that was seen clearly, there was no reason to stop at any particular n at all. The key-space stopped being a fixed number and became a configuration — as large as the dimension count you chose to use.

Mechanically, that meant rebuilding the Mandelbrot function itself to take a vector array rather than a fixed pair — n becomes a parameter, not a constant, and every dimension in the array is fully integrated into the function rather than bolted on beside it. The state-space inherits that same configurability directly: still 112 bits per dimension, but now multiplied by however many dimensions the array carries. 224 bits was never a ceiling in the fractal math itself — it was a ceiling in how many dimensions the standard Mandelbrot function uses.

Copyright © 2026 Wolfgang Flatow

To make that plain, without the fractal language: it's as if AES let you choose your own key size — not a fixed pick between 128, 192 and 256, but any size at all, simply by deciding how many dimensions to use. AES has no such lever. FES does, and it was never designed in from the start — it fell out the day I stopped trying to make two vectors do more, and asked instead why there had to be only two.

Of everything that had happened by then, this is the one I recognized the magnitude of, above all the rest. Not the fastest breakthrough, not the most dramatic — but the one that made "standard" the right word for what FES was becoming, rather than "technique." A technique solves one instance of a problem well. A standard has to hold across cases nobody's specified yet — including key sizes nobody's asked for, in configurations nobody's built. Once the key-space stopped being a number and became a parameter, FES was no longer just a scheme that worked. It was a scheme that could be asked to work at whatever scale a given case demanded, without anyone needing to redesign it first.

Performance

The first prototype's performance was ordinary, relative to AES. I hadn't optimized for speed, because speed had never been the point — the point had been whether the idea worked at all. It did, and for a long time that was enough.

I only began taking performance seriously once the quantum research had told me the project was worth pursuing properly. That changed the calculation entirely. Once I was looking at a genuine leap in state-space — a key-space that could scale far past what AES could offer, configurable rather than fixed — an ordinary performance result sitting next to that leap was a gap that needed closing.

The bottleneck, once I looked closely, it was plain enough: each fractal navigation iteration was only contributing a single byte to the fractal stream. That byte came from discarding the decimal point of the z value and taking it mod 256 — a small, cheap operation in itself, but it sat at the end of a much heavier chain of work. Each iteration had to run several trigonometric steps to map the new x,y location, then run the Mandelbrot function itself at that new location to extract z. All of that, for one byte. The iteration was non-trivial — the yield at the end of it trivial.

What emerged from sitting with that gap was a three-step innovation that lifted performance into the same ballpark as AES.

Step 1. It struck me that each z value was far richer than the single byte I'd been extracting from it — I was working in COM decimals, and a COM decimal carries far more underlying data than one mod-256 byte could ever reflect. So I tested copying the decimal z value directly to a byte array, using a memory copy rather than any decimal-point arithmetic. That single change delivered thirteen variable bytes out of a value I'd previously been squeezing down to one.

Copyright © 2026 Wolfgang Flatow

Step 2. Thirteen bytes wasn't the end of it, though — when I analyzed the byte patterns across large sets of navigated z values, all thirteen varied, but the variability wasn't even across the set. There was noticeably more variability toward the right-hand end of the thirteen bytes than the left. Left uncorrected, that unevenness would have found its way straight into the fractal stream. So I distributed it: a thirteen-byte shift register, rotated one byte per pass, maintained and XORed against the incoming decimal bytes. The shift register itself is what gets copied out to the fractal stream — the stream no longer touches the raw decimal bytes at all, only the distributed result sitting in the rotating register.

Step 3. The third step came from looking again at the multi-dimensional Mandelbrot function itself. Internally, it was already carrying a z() array — one z value per dimension — but the function had only ever been built to output the merged result, a single combined z, discarding the individual elements once they'd served their purpose. I parameterized that array for output instead of letting it collapse, and extracted the same thirteen bytes per z element, across every dimension in the array, rather than just once from the merged value.

Put together, the three steps changed the yield per iteration dramatically — from one byte, squeezed out of a discarded decimal point, to thirteen bytes per dimension, across however many dimensions the array carried, plus the merged z. With the minimum dimension count I settled on, eight, that's a substantial multiple of the original single-byte yield, for the same trigonometric and Mandelbrot work per iteration.

It also produced a balance I hadn't anticipated: as the dimension count goes up, fewer navigation iterations are needed to fill the same fractal stream length. More dimensions doesn't just mean more bytes per iteration — it means the whole stream gets there faster, because each iteration is now doing proportionally more of the work.

Core Overwrite

First Overwrite. The initial prototype kept the payload overwrite itself simple: a mod-256 addition for encryption, the matching subtraction for decryption. Simple as it was, I analyzed it properly rather than assuming its simplicity meant it was weak. It held up. There's no way to determine, from a sum alone, which two numbers were added to produce it — the ciphertext byte gives you nothing about the two addends that made it, only their combined result.

Even knowing something about the payload doesn't open much of a door. If you know the payload is a PDF, you can work out the stream bytes across the known header — that much structure is available to you. But past the header, there's nothing left to anchor against. The remaining stream bytes could be any value at all, and nothing in the ciphertext tells you which.

That property, first noticed here, at the simplest possible overwrite, was later analyzed properly and became the core of what made FES impenetrable — not a later addition bolted on for security, but something already sitting in the plainest version of the scheme, from the very first prototype.

Copyright © 2026 Wolfgang Flatow

Second Overwrite. At this point I decided the overwrite itself shouldn't be fixed. It should be optional and configurable — with a sensible default in place for anyone who didn't want to touch the setting at all, but nothing forcing a single method on everyone.

By this stage the prototype had a web interface, and the first pair of overwrite options I put in front of it as checkboxes were "Add" and "XOR." XOR was the obvious second choice to sit alongside addition — another two-way overwrite operation, just as reversible, just as simple to reason about.

Third Overwrite. Substitution — an S-box — was an obvious method to include next. What wasn't obvious was the best way to build it. The first implementation used a fixed, pre-mixed substitution array covering the 1–256 range, which then got shuffled further using the fractal stream itself. That worked reasonably well.

But when I looked closely at what that shuffle was actually doing, it was really only a rotation — not a full reordering, just values cycling around a fixed sequence. The version that made it into FES goes further: it runs the stream for an additional 256 decimal values — decimals, not bytes — and uses their raw values directly to re-order the pre-mixed array itself. The array doesn't just get rotated by the stream anymore. It gets genuinely reshuffled by it.

Secondary Overwrite

One or more of the core overwrites should be in use at all times — that was never optional in the way individual method choice was. What came later, on top of that, were additional bitwise overwrites, developed once the core set was already solid. These don't replace the core overwrites; they augment them, extending variability further for anyone who wants it. They're a security choice rather than a necessity — available for the cases that call for it, at the cost of greater processing overhead than the core set alone requires.

Chapter Four: Silos

Genesis

The Yamba prototype implements a fixed set of 2^{16} x,y vector pairs, generated by scanning a grid across the Mandelbrot fractal itself. Vectors that produced infinite z values were excluded deliberately — I wanted definite return values to work with, not values sitting at some max-iteration threshold standing in for infinity. A threshold value is a proxy; a definite return is the real thing, and the whole scheme was designed to work with the real thing.

The path from password to fractal portal ran through MDAC, in 128-bit chunks. Sixteen bits of each chunk select a vector pair out of the pre-scanned grid-scanned set. The remaining bits split in two — one half added to that vector's x, the other half added to its y — landing the portal at a location derived from, but never identical to, the selected grid point.

Copyright © 2026 Wolfgang Flatow

From Infinity Exclusion to Silo

Then came a real AHA, and it displaced the regular grid scan entirely. Instead of stepping methodically across the Mandelbrot set to assemble a fixed 2^{16} set of vectors, I realized I could assemble any unique set of 2^{16} x,y vectors at all — using GUIDs scaled to fractal vectors as inputs, exploiting the 29-decimal accuracy available in the decimal type, and testing each resulting x,y pair, discarding it if it produced an infinite z, exactly as before. Nothing about the exclusion rule changed. What changed was where the vectors came from.

I called the resulting set a Silo.

The Silo generation formula was tested and refined using a visual plot to confirm infinity exclusion and even distribution.

That one change moved the vector set from being a single, shared, grid-derived structure to something that could be generated fresh, uniquely, as many times as needed. And because each Silo is its own self-contained set of seed vectors, decryption using a given Silo is completely isolated to ciphertext that was generated using that same Silo. There's no shared grid underneath multiple encryptions to reason across — each Silo stands alone.

Since GUIDs can generate 2^{128} unique and distinct inputs, an infinite number of unique Silos could be produced from this method by combining random sets of 2^{16} x 2 GUIDS to generate a Silo — not a large fixed library of them, but a genuinely open-ended supply.

Silos Solve Encryption Monoculture

There's a risk in cryptography that sits above any single key, any single message, any single attack — the risk of monoculture. AES is one algorithm, one fixed structure, running the same way in every deployment, everywhere in the world. That uniformity is a large part of why it's trusted: it's been studied exhaustively, by more eyes than almost anything else in the field, and nothing has broken it. But the same uniformity means that any structural weakness discovered against AES — QKE, or anything else, present or future — doesn't threaten one system. It threatens all of them, everywhere, at once, because they're all running the identical underlying structure.

Silos change that calculation from the ground up. Every Silo is its own unique set of seed vectors, generated independently, isolated from every other Silo that's ever been produced or ever will be. A weakness found against one Silo's specific geometry doesn't transfer to another's, because there's no shared structure underneath them to transfer through — each one is its own foundation, not an instance of a shared one. Break the reasoning behind one Silo, if such a thing were even possible, and you haven't broken FES. You've broken exactly one Silo, out of an effectively infinite supply of others that owe it nothing.

Copyright © 2026 Wolfgang Flatow

That's not a wholly separate kind of security from key secrecy — it feeds directly into it, and dramatically so. With a shared structure like AES, a single key compromise in one location doesn't stay contained to that location. The same fixed algorithm sits underneath every other deployment too, so whatever a compromise reveals about how the structure behaves extends outward, indefinitely, to every other instance running the same thing. Silos cut that off at the source. A compromise against one Silo's geometry has nowhere to travel to, because there's no shared structure underneath other deployments for it to extend into. Key secrecy still matters — a compromised key is still a real loss — but Silos stop that loss from radiating outward into every other user of the standard, the way it can when everyone shares one foundation. AES was never built to need that property, because nobody building it in the 1990s was worried about a monoculture threat model at that scale. FES has it, not because I set out to solve monoculture specifically, but because the Silo insight — assemble a unique vector set rather than share one — happened to dissolve the problem as a side effect of solving something else entirely.

Private Cryptographic Foundations

Because Silos can be generated on an as-needed basis, that isolation isn't something I have to manage centrally on anyone's behalf. Any organization, agency, or government can generate its own Silos directly, matching them to whatever compartments already exist inside their own structure — need-to-know boundaries that were already there, now given a genuine cryptographic wall to match, rather than a shared one everyone quietly depends on together.

The public FES web interface has included Silos since 2023. I'd put that plainly as one of the more remarkable developments in the history of cryptography — not because of any claim I'm making about it, but because of what it quietly hands to anyone who uses it: the ability to walk away from monoculture entirely, generate a genuinely private cryptographic foundation on demand, and never share it with anyone who wasn't meant to have it.

Chapter Five: Fractal OTP

Meeting Shannon's OTP Requirements

Shannon's One-Time Pad has three requirements. Meet all three, and the result is proven, mathematically, to be unbreakable — not "very hard to break," but information-theoretically incapable of it, regardless of the attacker's computing power, then or ever.

1. **The key must be truly random.** Not pseudo-random, not derived from a pattern that could in principle be reconstructed — genuinely unpredictable, with no structure an attacker could exploit even in theory.
2. **The key must be at least as long as the message.** Every bit of plaintext needs its own bit of key standing behind it. Reuse any portion of the key across more plaintext than it was generated for, and the guarantee collapses.

Copyright © 2026 Wolfgang Flatow

3. **The key must never be reused.** Use the same key twice, on two different messages, and an attacker who has both ciphertexts gains real leverage against both — the perfect-secrecy guarantee depends entirely on the key being spent once and never again.

Meeting all three in practice is notoriously difficult, which is exactly why the classical one-time pad has stayed a theoretical benchmark more than a working system. True randomness at scale is expensive to generate and awkward to distribute. A key as long as the message means the key-distribution problem is exactly as large as the message-distribution problem, which defeats much of the point. And "never reused" demands a discipline that's easy to state and hard to enforce once a system is in real use, by real people, under real pressure.

FES's fractal stream, once I looked at it against these three requirements directly rather than assuming it would simply qualify, turned out to satisfy all three in a practical, working framework — not as a theoretical ideal sitting apart from any real implementation, but as something the scheme actually does, every time it runs.

Verifying FES OTP Requirements

Each of Shannon's three requirements is its own separate challenge, with its own measure and its own kind of evidence. They don't come as a package — each one had to be checked on its own terms.

OTP#2 — key length matching the message is the most straightforward of the three, because it was never really in question. FES always generates a fractal stream matched to the payload's own length. If more than one pass is selected, the stream simply runs to payload length times passes. There's no case where the stream falls short of the message it's covering.

OTP#1 — true randomness is a different matter entirely, and it took real work to move from intuition to evidence. At first, my confidence in it was intuitive — built on what I already understood about fractal navigation, and on the proven, infinite complexity of the Mandelbrot set itself. That was a reasonable starting confidence, but it wasn't proof.

So in 2025 I built a publicly accessible tool, FES Gauge, specifically to put that confidence to the test. It runs a range of utilities against the fractal stream directly — randomness measures, compressibility tests among them — open for anyone to run themselves rather than take my word for. The results have been consistently impressive, regardless of how FES is configured. No configuration I've tested has produced a stream that falls short.

What makes that result worth pausing on is what's absent from the mechanism producing it: there is no random function anywhere in FES. None. The entire stream comes out of deterministic fractal navigation — the same Mandelbrot math, run the same way, every time. And yet it measures as close to perfectly unpredictable as randomness testing can show, with effectively zero compressibility, consistently. That's not a small thing to find. A system with no randomness function in it, producing output indistinguishable from true randomness under

Copyright © 2026 Wolfgang Flatow

direct testing, satisfies OTP#1 in practice — not by simulating randomness, but by generating something that holds up against every test built to catch the difference.

OTP#3 — never reusing the key is the hardest of the three, and it's a genuinely different kind of problem from the other two. Reuse a password, and you reuse the same fractal stream behind it. Without addressing that, FES is still impenetrable from the standpoint of ciphertext knowledge alone — nothing in a single ciphertext gives an attacker traction, regardless of reuse. But that's not the same as meeting OTP#3, which is a stricter, structural requirement about the key itself, not just about what a single piece of ciphertext reveals.

Working through that gap is what led to the Fractal OTP offset: a two-stage process for portal discovery. The password and Silo generate the multi-dimensional portal exactly as before — nothing changes there. But then the resulting fractal stream itself is traversed directly using the Fractal OTP value, where its values are used to shift that portal to a new location within the multi-dimensional space, before the real encryption stream is generated from that shifted point.

That two-stage structure means the same password can be reused safely, as long as something else — a session ID, a file path, anything that varies — is folded in to produce a distinct offset each time. The password stays constant and memorable; the fractal stream it ultimately produces never repeats.

Fractal OTP offset has been integrated into FES since 2023.

Chapter Six: Overwrite Passes

Fractal Stream Continuation

Passes had already entered this story earlier, in passing — the fractal stream running to payload length times passes, when more than one pass is selected. It's worth stopping here and looking at what that actually means, because it's a simpler idea than it might sound, and the simplicity is the point.

A single pass generates one fractal stream, exactly matched to the payload, and consumes it once, through whatever overwrite methods are configured. A second pass doesn't restart anything — it doesn't reset the navigation, regenerate a new portal, or touch the Silo again. It continues the same fractal stream forward from wherever the first pass left off, and applies a second full round of overwrite against the payload using that continuation.

Each additional pass is more of the same continuous traversal, never a repeat of ground already covered. The stream never loops back on itself and never reuses a stretch it's already produced. That matters for the same reason OTP#3 mattered a chapter ago: reuse is the thing that opens a door, and continuation without reuse is what keeps that door shut, no matter how many passes are configured.

Copyright © 2026 Wolfgang Flatow

What passes buy you, then, isn't variety for its own sake — it's depth. Each additional pass layers another full application of the configured overwrites on top of the ones before it, using stream material that has never been used anywhere else, in this payload or any other. More passes mean more work for anyone attacking the result, at the direct cost of more work for the machine doing the encrypting. That trade is left entirely configurable — nothing about FES insists on a particular number of passes, because different cases call for different points on that trade.

Contrast to AES multi-pass

AES's own multi-round structure looks superficially similar, and it's worth being precise about why it isn't the same thing. AES already runs multiple rounds internally — ten, twelve, or fourteen, depending on key size — each round applying the same fixed transformation, driven by a key schedule expanded once, up front, from the original key. Every round draws on material derived from that same single key. There's no fresh entropy entering the process round to round; the rounds are repetitions of a fixed structure, not a continuing stream of new material. And if you go further and encrypt with AES twice, using two different keys, you don't get double the effective security — known attacks against multiple encryption (meet-in-the-middle among them) erode a meaningful share of the gain you'd naively expect from stacking two independent keys.

FES's passes don't share that ceiling, because they're not repeating a fixed transformation on a static key schedule — each pass draws on genuinely new stream material, continued forward rather than recycled, with nothing to meet in the middle of because there was never a fixed structure sitting underneath multiple applications to begin with. Where AES's rounds are the same door, opened the same way, however many times you knock, FES's passes are new ground, every time.

Overwrite Options Order

Whatever set of overwrite options is configured — Scramble included — repeat on every pass, each time drawing on the fresh stream material that pass provides. Nothing configured gets a single application and then steps aside for the rest.

The order they run in is fixed for encryption, but reversed for decryption. That symmetry sounds simple to state, but it isn't simple to build. Reversing a sequence of overwrites correctly means each one has to be individually invertible, in exactly the right order, working backward through whatever was configured. The Fractal S-Box, in particular, demands specific handling in reverse — undoing a genuine reshuffle of the substitution array is a different, more exacting operation than undoing the addition or XOR steps sitting alongside it, and it has to land exactly right, every time, or decryption doesn't recover the payload at all. There's no partial credit in a reversal like this — it either lines up completely, pass by pass and option by option, or the output is simply wrong.

Copyright © 2026 Wolfgang Flatow

Chapter Seven: FES Matured

New Cryptographic Art & Science

That's the story of how FES became the standard you see today — not as a single arrival, but as one continuous run of invention, innovation, research, and bloody-minded, persistent curiosity, each piece resting on the one before it.

I want to close by being honest about how little enthusiasm was in the room at the start. The first prototype didn't excite me the way this account might suggest it should have. "Interesting new encryption" was about the size of it. I put it aside. AES was the NIST standard, studied and trusted by the whole world, and I never thought for a moment that what I'd built belonged anywhere near that league. There was no reason to think otherwise, and I didn't.

What changed everything was learning about quantum computers. I gave a year to it — everything I could absorb about qubits, entanglement, superposition — and it remains one of the most remarkable stretches of learning I've had. As the real power of superposition became clear to me — a key-space entangled and accessed simultaneously, all of it, at once, rather than searched through one guess at a time — I understood that AES was facing something genuinely new. So the plain question followed naturally: how would FES fare against the same thing?

The answer surprised me. FES resisted quantum computing in ways I hadn't anticipated when I first asked the question, and Shannon's OTP requirements turned out to be sitting right at the center of why.

FES discards the password material the moment the fractal portal is discovered — irreversibly, at the very first stage, before the payload is ever touched. From there, the fractal stream takes over the entire payload, each navigation iteration also irreversible, and that has a striking consequence: every possible bit combination in the resulting ciphertext is a viable decrypt, including every sensible-looking one among them. There's no privileged answer sitting inside the ciphertext for anything — quantum or classical — to key on. AES, by contrast, has exactly one sensible result waiting to be recognized. That single difference is the whole of it.

That surprise — finding real resistance where I hadn't expected to look for it, and finding the reason for it in Shannon's own standard rather than in anything I'd claimed for myself — is what kept me going. It's the reason the R&D didn't stop once curiosity had been satisfied, the way it had after Yamba the first time. It's the reason FES kept developing, year after year, until it became the standard sitting in front of you now.

Discover FES: https://portalz.solutions

Copyright © 2026 Wolfgang Flatow