Overview & Concepts

Table of Contents

---

1. Introduction

1.1 What This System Is
1.2 What This System Is Not
1.3 Intended Use Cases

2. System Overview

2.1 High-Level Architecture
2.2 Component Roles

3. Core Data Structures

3.1 The Lexicon
3.2 Chunks
3.3 Word.Iteration Schema
3.4 Mnemonic as Transformation Assembly
3.5 Indices as State Selectors
3.6 Randomness Source
3.7 Determinism vs Variability

4. Active Key Construction

4.1 Mnemonic Parsing Rules
4.2 Chunk Selection Logic
4.3 Resulting Active Key

5. Transformation Model

5.1 Pass-Based Processing
5.2 Index Mechanics
5.3 Directional Agnosticism

6. Determinism & Behavior

6.1 Deterministic Inputs → Deterministic Outputs
6.2 Effects of Changing:
6.3 Modulo Normalization
6.4 Behavioral Implications

7. Failure Modes & Verification

7.1 Silent Failure Modes
7.2 False Confidence Risks
7.3 Verification Strategies
7.4 Irreversibility Conditions (optional but strong)

8 Comparison to One-Time Pad Systems

8.1 Similarities

8.2 Key Differences

8.3 Interpretation Guidelines

9. Design Evolution (Vizini → Current)

9.1 Key Differences from V1

9.2 Rationale for Changes

10. Terminology

10.1 Canonical Definitions

10.2 Notes on Non-Standard Terminology

---

1. Introduction

1.1 What This System Is

Vizini v2 is a deterministic, lexicon-driven data transformation system that derives its behavior from a combination of structured data and user-provided inputs.

Rather than using a single static key, Vizini v2 generates a transformation space based on:

- A shared Lexicon (VLex database)
- A mnemonic phrase
- User-defined indices and parameters

Given the same inputs, the system will always produce the same output, making it reversible when all required inputs are reproduced exactly.

The system is designed to explore alternative approaches to data transformation, key derivation, and entropy utilization. It emphasizes flexibility and composability over strict adherence to established cryptographic standards.

1.2 What This System Is Not

Vizini v2 is not a formally validated cryptographic system and should not be treated as a substitute for established encryption standards.

Specifically:

- It has not undergone formal cryptographic analysis or peer review
- It does not guarantee confidentiality, integrity, or authenticity
- It does not include built-in mechanisms for verification or tamper detection
- It is not designed for real-time communication, streaming, or network protocols
- It should not be assumed secure against adversarial analysis

While the system may exhibit encryption-like behavior, its security properties are not formally defined.

1.3 Intended Use Cases

Vizini v2 is best suited for controlled, experimental, or exploratory applications where its behavior can be fully understood and externally validated.

Appropriate use cases include:

- Experimentation with lexicon-based transformation systems
- Obfuscation where strict security guarantees are not required
- Educational exploration of deterministic transformation models

- Offline workflows where both input parameters and outputs can be verified independently

It may also be useful in scenarios where:

- Portability is important (e.g., removable media workflows)
- Reproducibility is required across isolated systems
- Custom transformation behavior is desirable
- Plausible Deniability is a valuable security strategy

Vizini v2 is not recommended for:

- Protecting sensitive or high-value data without additional safeguards
- Situations requiring strong, proven cryptographic guarantees
- Systems that require built-in validation, authentication, or error detection

2. System Overview

2.1 High-Level Architecture

Vizini v2 is composed of three loosely coupled components that operate on shared data structures rather than direct inter-process communication.

At a high level, the system follows a produce → analyze → apply workflow:

- A Lexicon is generated or modified
- The Lexicon can be inspected and analyzed
- The Lexicon is used as part of a deterministic transformation process

All components interact through portable, file-based artifacts:

- VLex databases (SQLite-based Lexicons)

- User-provided mnemonic input
- Transformation parameters (indices, offsets, iteration depth, etc.)

There is no requirement for background services, persistent daemons, or network connectivity. Each component can operate independently, provided it has access to the required inputs. There is no shared runtime state between components beyond these artifacts.

This architecture emphasizes:

- Portability - all required data can be moved between systems
- Determinism - identical inputs produce identical outputs
- Separation of concerns - each tool has a focused role

2.2 Component Roles

Scribe (Lexicon Creation & Modification)

Scribe is responsible for generating and modifying Lexicons (VLex databases), which serve as the foundational data structure for all transformations.

Its primary responsibilities include:

- Generating high-entropy Lexicons using the system CSPRNG
- Shuffling and structuring internal mappings (e.g., S-box construction)
- Allowing manual or semi-manual Lexicon creation (if desired)
- Persisting Lexicons as portable SQLite database files

The quality and structure of the Lexicon directly influence the behavior of the transformation process.

LAT / Audit (Lexicon Inspection & Analysis)

LAT (Lexicon Analysis Tool) provides visibility into the structure and characteristics of a Lexicon.

It is used to:

- Analyze word distribution and weighting
- Evaluate mnemonic interaction depth and coverage
- Inspect iteration behavior and structural patterns
- Identify potential anomalies or unintended structure

LAT does not modify the Lexicon-it is strictly an analysis and auditing tool.

This component is especially important for understanding how a given Lexicon may behave when used in transformation.

Vizini2 the Cipher Engine **(Data Transformation)**

The Vizini2 Engine performs the actual data transformation process.

Using:

- A Lexicon (VLex)
- A mnemonic phrase
- User-defined indices and parameters

...the engine derives a deterministic transformation structure and applies it to input data.

Key characteristics:

- Fully deterministic given identical inputs
- Produces output for any valid input (no inherent failure state)
- Fully reversible when all inputs are correctly reproduced
- Does not include built-in validation or integrity checking

The Engine is where all components converge, turning structured inputs into transformed output data.

3. Core Data Structures

3.1 The Lexicon

The Lexicon (VLex) is the central data structure in Vizini v2. It defines the transformation space used by the system.

Lexicons are stored as SQLite databases, providing portability, reliability, and a well-understood storage format.

A typical Lexicon contains:

- A metadata table describing global properties (Title of Lexicon, Who made it, and Notes from the Lexicon Author)
- A chunks table containing the core transformation structures

The Lexicon is both a dataset and is a structured mapping system that directly influences how input data is transformed.

3.2 Chunks

Chunks are the fundamental building blocks within a Lexicon.

Definition

A Chunk represents a grouped unit of transformation data, consisting of one or more substitution mappings (S-boxes).

Structure

Each Chunk contains:

- One or more S-boxes (substitution tables)
- An associated identifier (e.g., word + iteration)
- Internal ordering that affects traversal and application

Chunks are designed to be composable and iterable, allowing complex transformation behavior to emerge from relatively simple components.

Weight

Weight refers to the number of S-boxes contained within a Chunk.

- Higher weight = more s-boxes per Chunk = increases transformation complexity and diffusion
- Lower weight = fewer s-boxes per Chunk = decreases transformation complexity and diffusion

Weight influences transformation complexity and diffusion, but does not increase the entropy of the underlying source data.

3.3 Word.Iteration Schema

Chunks are identified using a structured naming convention:

```
word.iteration
```

Examples:

- alpha.1
- alpha.2
- beta.1

Naming Convention

- The word component groups related Chunks
- The iteration component distinguishes multiple variants associated with that word

Depth

Depth refers to how many unique iterations exist for a given word within the Lexicon.

- Higher depth = more unique variants available
- Lower depth = fewer unique variants before reuse occurs

Depth defines the boundary at which repeated use of a word in a mnemonic may begin to reuse underlying transformation structures.

Iteration Behavior

Iteration behavior describes how repeated occurrences of a word in a mnemonic are resolved.

Depending on the Lexicon:

- Each occurrence may map to a unique iteration (up to depth)
- Beyond depth, iterations may repeat or cycle
- Different iterations of the same word are expected to produce distinct transformation effects

This schema allows a single mnemonic element to expand into a structured sequence of transformation steps.

3.4 Mnemonic as Transformation Assembly

The mnemonic phrase acts as the primary driver for selecting and ordering Chunks during transformation.

Rather than functioning as a traditional cryptographic key, the mnemonic:

- Selects which word groups are used
- Determines the order of Chunk inclusion
- Expands deterministically into a sequence of S-boxes based on iteration rules

Each mnemonic element found in the Lexicon maps to one Chunk via the word.iteration schema, contributing its weight (S-box count) to the Active Key.

Mnemonic elements not present in the Lexicon are ignored.

The mnemonic therefore acts as a deterministic assembly layer, shaping how the Lexicon is applied.

3.5 Indices as State Selectors

Indices define the starting state within the assembled transformation sequence (Active Key).

They are used to:

- Select the initial position within the S-box sequence
- Determine offsets into transformation structures
- Influence traversal starting points

While the mnemonic defines what is used, indices define where execution begins.

Small changes in indices can result in entirely different transformation paths through the Active Key.

3.6 Randomness Source

Scribe uses the operating system's cryptographically secure pseudorandom number generator (CSPRNG) to generate S-box permutations.

Each S-box is shuffled four times to promote high-quality distribution.

While it is theoretically possible to construct a Lexicon using true random sources, the use of a CSPRNG is considered sufficient for practical purposes, as it provides:

- High entropy
- Non-predictable output
- Strong statistical properties

The system’s behavior is therefore not limited by randomness quality, but by how that randomness is structured and applied.

3.7 Determinism vs Variability

Vizini v2 operates at the intersection of determinism and variability.

Determinism

Given identical inputs:

- Lexicon
- Mnemonic
- Indices and parameters

...the system will always produce the same output.

This ensures reproducibility and reversibility under correct conditions.

Variability

Changing any component-even slightly-can produce drastically different outputs:

- Different Lexicon = entirely different transformation space
- Different mnemonic = different selection and ordering of S-boxes
- Different indices = different starting state

This creates a system where:

- Behavior is fully deterministic
- Outcomes are highly sensitive to input variation

This duality is central to how Vizini v2 functions.

4. Active Key Construction

4.1 Mnemonic Parsing Rules

The mnemonic is first normalized and tokenized before any Chunk selection occurs. This ensures consistent behavior regardless of formatting differences.

Lowercasing

All mnemonic input is normalized to lowercase prior to processing.

This avoids case-sensitivity issues and ensures that:

- Alpha, ALPHA, and alpha are treated identically
- Users are not required to remember exact casing

This normalization improves usability without affecting deterministic behavior.

Delimiters

Mnemonic input is tokenized using non-alphanumeric characters as delimiters.

This includes:

- Whitespace
- Punctuation
- Symbols

As a result, the following are split into multiple tokens:

- Contractions (don't → don, t)
- Possessives (user's → user, s)
- Hyphenated words (high-speed → high, speed)
- Numeric groupings (123-456 → 123, 456)

This behavior may be non-intuitive in some cases, but it ensures consistent and implementation-simple tokenization.

Ignored Tokens

Tokens that do not exist in the Lexicon are silently ignored.

This allows users to include:

- Filler words for readability
- Natural language phrasing
- Structurally helpful but non-functional words

For example, a mnemonic may include words purely to improve memorability or linguistic flow, even if they do not contribute to the transformation.

Ignored tokens do not affect the resulting Active Key.

4.2 Chunk Selection Logic

Once tokenized, each valid token is mapped to a Chunk using the word.iteration schema.

First Occurrence vs Repeated Words

Each occurrence of a word in the mnemonic is processed independently.

- The first occurrence maps to the first available iteration (e.g., word.1)
- Subsequent occurrences advance through available iterations

Example:

alpha alpha alpha

May resolve to:

- alpha.1
- alpha.2
- alpha.3

This allows repeated words to expand into distinct transformation components.

Iteration Exhaustion Behavior

When the number of occurrences of a word exceeds its available depth, iteration selection wraps using modulo behavior.

Example (depth = 2):

alpha alpha alpha alpha

Resolves to:

- alpha.1
- alpha.2
- alpha.1
- alpha.2

This introduces reuse of underlying transformation structures.

As repetition increases beyond depth, the transformation begins to reuse previously selected Chunks, which may reduce effective variation.

4.3 Resulting Active Key

After processing all valid tokens, the system constructs a linear sequence of S-boxes derived from the selected Chunks.

This sequence is referred to as the Active Key (colloquially), and represents the full transformation structure applied to the data.

Ordering Guarantees

The Active Key preserves the exact order of mnemonic-derived Chunk selection.

This means (assuming the same Lexicon is used):

- Identical mnemonic sequences produce identical Chunk sequences
- Repeating a sequence of words produces a repeated transformation structure

Example:

alpha beta alpha beta

Will always produce the same ordered sequence of Chunks and S-boxes.

There is no internal randomization at this stage - ordering is strictly deterministic.

Total Weight

The total weight of the Active Key is the sum of the weights of all selected Chunks.

This determines:

The total number of S-boxes in the transformation sequence

The overall length and complexity of the transformation path

Higher total weight results in:

- More transformation steps
- Greater diffusion across the data

Lower total weight results in:

- Shorter transformation sequences
- Simpler transformation behavior

The Active Key is fully determined before transformation begins and remains static for the duration of the operation.

5. Transformation Model

5.1 Pass-Based Processing

Vizini v2 processes data using a two-pass transformation model, where each pass operates over the same Active Key but may differ in direction and initialization.

Forward Pass

The forward pass:

- Uses the initial index labeled “Pass 1” in the Vizini2 interface
- Processes data in the defined traversal order of the Active Key from the begining of the source material to the end.

- Applies S-box transformations sequentially based on index progression

This is typically associated with “encryption,” though the system itself does not enforce semantic labeling.

Reverse Pass

The reverse pass:

- Uses the initial index labeled “Pass 2” in the Vizini2 interface
- Processes data in the defined traversal order of the Active Key from the end of the source material to the begining.
- Employs an inverse S-box lookup function

Despite being referred to as “Pass 2,” this pass is still logically the second (backward) stage of transformation-even when the overall process is inverted.

Pass Inversion Behavior

During reverse operations (e.g., decryption):

- The order of passes is inverted
- Each S-box operation is replaced with its inverse

However, the labels “Pass 1” and “Pass 2” remain tied to their directional correlation, not execution order.

This ensures consistent configuration regardless of operation direction.

5.2 Index Mechanics

Indices define the evolving state of traversal through the Active Key during transformation.

Initial Index Inputs

Each pass begins with its own user-defined initial index:

- Pass 1 = Forward pass starting index
- Pass 2 = Reverse pass starting index

These indices determine the initial position within the Active Key's S-box sequence.

Index Shifting Behavior (Ciphertext-Driven)

After each byte is processed, the active index is updated based on the value of the resulting ciphertext byte.

Specifically:

- The index is incremented by the numeric value of the processed ciphertext byte
- If the index exceeds the bounds of the Active Key (total weight), it is wrapped using modulo arithmetic

```
new_index = (current_index + ciphertext_byte) mod total_weight
```

This creates a data-dependent traversal path, where each output byte influences the position of the next transformation step.

Implications of Index Shifting

- Identical plaintext processed with identical inputs will produce identical output (determinism)
- Small changes in input propagate through index shifts, affecting all subsequent transformations
- The transformation path is stateful and self-influencing

5.3 Directional Agnosticism

Vizini v2 does not fundamentally distinguish between encryption and decryption as separate algorithms.

Instead, both operations are symmetric transformations with inverted parameters.

Encrypt vs Decrypt Equivalence

- Encryption and decryption use the same core transformation logic
- Decryption reverses:
    - Pass order
    - S-box mappings (forward vs inverse)

When all inputs are correctly reproduced, the transformation is fully reversible.

Performance Characteristics

Decryption may be marginally slower than encryption due to use of inverse S-box lookups instead of direct indexed access

This difference is typically minimal but can become noticeable with large datasets or high-weight Active Keys.

6. Determinism & Behavior

6.1 Deterministic Inputs → Deterministic Outputs

Vizini v2 is fully deterministic.

Given identical inputs:

- Lexicon
- Mnemonic
- Starting indices and parameters

...the system will always produce identical output.

There is no internal randomness during transformation. All variability is introduced exclusively through input differences.

This determinism ensures:

- Reproducibility across systems
- Consistent forward and reverse transformation
- Reliable behavior in controlled environments

6.2 Effects of Changing Inputs

Small changes to any input component can produce large and unpredictable differences in output.

Mnemonic Changes

Changing the mnemonic:

- Alters Chunk selection
- Changes the ordering of S-boxes
- Modifies the structure of the Active Key

Even minor changes (e.g., a single word) can produce entirely different transformation behavior.

Lexicon Changes

Changing the Lexicon:

- Replaces the entire transformation space
- Alters all S-box mappings
- Invalidates compatibility with previous outputs

Even if the mnemonic and indices remain the same, a different Lexicon will produce completely different results.

Starting Index Changes

Changing starting indices:

- Alters the initial position within the Active Key
- Changes the traversal path through S-boxes

- Affects all subsequent index shifts

Because index shifting is ciphertext-driven, even small index changes propagate through the entire transformation.

6.3 Modulo Normalization

Vizini v2 uses modulo normalization to ensure all index operations remain within valid bounds.

Index Wrapping

When an index exceeds the bounds of the Active Key (total weight), it is wrapped using modulo arithmetic:

```
index = index mod total_weight
```

This guarantees:

- All index values remain valid
- No out-of-bounds access occurs
- Traversal continues seamlessly regardless of shift magnitude

Overflow Handling

Index growth is unbounded in theory but constrained in practice through normalization.

Because index updates are based on ciphertext byte values:

- Large shifts are possible
- All shifts are safely normalized within the Active Key range

This ensures consistent behavior regardless of input size or content.

6.4 Behavioral Implications

The combination of determinism and input sensitivity produces several important characteristics:

- Identical inputs always produce identical outputs
- Small input changes produce large output differences
- Transformation behavior is entirely input-driven
- There is no internal mechanism to verify correctness

Because any valid input produces an output:

- Incorrect inputs will not fail visibly
- Outputs may appear valid even when incorrect
- External validation is required when correctness matters

(See Section 7.3 for recommended validation strategies.)

7. Failure Modes & Verification

7.1 Silent Failure Modes

Vizini v2 does not provide internal validation of correctness.

Any variation in input-including changes to the Lexicon, mnemonic, or starting indices-will produce a valid transformation output, regardless of correctness.

There are no errors, warnings, or indicators of misuse.

Because the transformation engine operates strictly on a “data in → data out” model, it is also possible to apply additional transformation layers without restriction. This flexibility increases composability, but does not provide any inherent validation.

7.2 False Confidence Risks

The absence of internal validation introduces the risk of false confidence.

- No checksum, marker, or structural validation is performed

- Incorrect settings may produce output that appears valid or readable
- There is no distinction between correct and incorrect results at the system level

As a result, users may incorrectly assume that a transformation was successful when it was not.

This risk is especially significant when working with human-readable data, where incorrect outputs may still appear plausible.

7.3 Verification Strategies

To ensure correctness, external verification mechanisms must be used.

Common strategies include:

- Known plaintext markers - Embedding recognizable data (e.g., fixed headers or phrases) to confirm successful reversal
- Embedded structure - Including magic bytes, version identifiers, or structured headers within the data
- External hashing / checksums - Verifying output integrity using hashes (e.g., SHA-family) or checksums calculated prior to transformation
- Redundant encoding - Encoding the same data multiple times or in multiple forms to allow consistency checks

These approaches allow users to distinguish between correct and incorrect outputs despite the system's lack of internal validation.

7.4 Irreversibility Conditions

Certain failure conditions result in practical irreversibility.

Loss of Lexicon

- The Lexicon defines the entire transformation space
- Without it, reconstruction is not feasible

Recovery probability is effectively zero.

Loss of Mnemonic

- The mnemonic determines Active Key construction
- Without it, reconstruction requires brute-force search

While tools such as LAT may assist in analysis, the search space remains extremely large and recovery is unlikely in practice.

Parameter Mismatch

- Incorrect starting indices alter traversal through the Active Key
- Even correct mnemonic and Lexicon combinations will fail if parameters differ

The total number of possible index combinations is bounded by the square of the Active Key weight:

```
total_combinations ≈ (total_weight)²
```

Even moderate weights result in impractically large search spaces for brute-force recovery.

8. Comparison to One-Time Pad Systems

8.1 Similarities

This system shares some conceptual similarities with one-time pad (OTP) encryption:

- Output is highly dependent on a shared entropy source
- Correct decoding requires exact coordination of parameters
- The space of possible outputs is extremely large

8.2 Key Differences

However, it differs in several important ways:

- The entropy source is structured and reusable rather than truly random and single-use
- The transformation process is stateful
- No formal proof of perfect secrecy exists
- The system does not guarantee indistinguishability of outputs

8.3 Interpretation Guidelines

As a result, this system should not be considered equivalent to a one-time pad.

9. Design Evolution (Vizini → Current)

Vizini v1 was developed as a proof-of-concept to validate the core transformation model: an indexed collection of S-boxes, with selection driven by feedback from the transformation process.

Vizini v2 builds on this foundation by introducing dynamic key construction, improved randomness, and a more structured data model. These changes shift the system from a fixed experimental design to a flexible and user-driven transformation framework.

9.1 Key Differences from V1

- Dynamic Index Construction
    - v1 used a fixed index of exactly 256 S-boxes.
    - v2 constructs the index dynamically using a Lexicon and mnemonic, allowing the total size (weight) to vary based on user input.
- Randomness Source
    - v1 used standard pseudorandom number generators (PRNGs).

    - v2 uses the operating system's cryptographically secure pseudorandom number generator (CSPRNG).
- Transformation Pass Model
    - v1 performed a single forward pass.
    - v2 performs two passes in opposing directions.
- Key Material Integration
    - v1 optionally applied a looped password XOR overlay.
    - v2 incorporates the mnemonic directly into Active Key construction as a required component.
- Storage Format
    - v1 stored S-boxes as raw serialized data.
    - v2 uses a structured SQLite database.
- Data Model Expansion
    - v1 stored only S-boxes.
    - v2 stores S-boxes along with metadata and organizational structures.
- Development Environment
    - v1 was developed on Windows using Delphi.
    - v2 is developed on Linux using Lazarus.

9.2 Rationale for Changes

Dynamic Index Construction

- Enables user-defined transformation complexity and structure.
- This increases variability and eliminates the predictability of a fixed S-box set, while allowing the Active Key to scale with mnemonic content.

Improved Randomness Source

- Using a CSPRNG improves entropy quality and reduces predictability in S-box generation.
- This strengthens the statistical properties of the transformation space without requiring external randomness sources.

Double-Pass Transformation

- Applying forward and reverse passes increases diffusion and interdependence between data elements.
- This makes the transformation more sensitive to input changes and reduces simple structural artifacts from a single pass.

Mnemonic as Core Key Material

- Moving from an optional XOR overlay to a required mnemonic shifts the system from additive masking to structural keying.
- The mnemonic now directly defines how the transformation operates, rather than acting as a secondary layer.

Structured Storage (SQLite)

- A database-backed design enables efficient storage, retrieval, and modification of S-box data.
- It also allows for scalability, persistence, and compatibility with analysis tools.

Metadata Integration

- Storing metadata alongside S-boxes enables organization, labeling, and analytical inspection (e.g., via LAT).
- This improves usability and supports more advanced workflows.

Platform & Toolchain Shift

- Moving to Linux and Lazarus maintains language continuity while improving portability and development flexibility.
- It also aligns the system with environments more commonly used for infrastructure, tooling, and experimentation.

Vizini v2 shifts the system from a fixed transformation mechanism to a user-defined transformation space.

10. Terminology

10.1 Canonical Definitions

General Cryptographic Terms

- Substitution Cipher : A method of encoding where each unit of data (typically a byte) is replaced with another value according to a defined mapping.
- S-Box (Substitution Box) : An indexed lookup table that defines a substitution mapping from input values to output values. In bijective systems, each value appears exactly once.
- Plaintext : The original, untransformed data.
- Ciphertext : The transformed output produced by the system.
- Bijective Mapping : A one-to-one correspondence between input and output values, ensuring perfect reversibility without data loss.
- Entropy : A measure of unpredictability or randomness. Higher entropy generally increases resistance to prediction or analysis.
- Brute Force : An attack method that attempts all possible input combinations (e.g., mnemonics, indices) to recover a correct transformation.

Technical Implementation Terms

- SQLite : A lightweight, serverless, file-based relational database engine used for structured data storage.
- BLOB (Binary Large Object) : A data type used to store raw binary data, such as S-box contents.
- Strip (Command) : A build process step that removes debugging symbols from compiled binaries to reduce file size.

10.2 Notes on Non-Standard Terminology

Project-Specific Terms

- VLex (Vizini Lexicon) : A portable SQLite database generated by Scribe. It contains collections of S-boxes grouped by word and iteration, along with associated metadata, and serves as the source material for Active Key construction.
- Mnemonic : A human-readable phrase used to select and assemble data from the VLex into an Active Key.
- Chunk : A collection of one or more S-boxes associated with a specific word and iteration. Chunks are the fundamental building blocks used during Active Key construction.
- Iteration (Depth) : The number of distinct versions of a word stored in the VLex. Each iteration corresponds to a unique Chunk. Higher depth increases variability when words repeat.
- Weight : The number of S-boxes contained within a single Chunk. Weight determines how much influence a word iteration has on the Active Key and affects total traversal length.
- Active Key : A linear sequence of S-boxes assembled at runtime from the VLex using the mnemonic. This structure defines the transformation behavior.

- Index (Traversal Index) : The current position within the Active Key used to select the active S-box during transformation. The index is updated dynamically based on ciphertext-driven shifting.
- Total Weight : The total number of S-boxes in the Active Key. This defines the bounds for index traversal and modulo normalization.
- Modulo Normalization : The process of constraining index values within valid bounds using modulo arithmetic.
- Pass (Transformation Pass) : A complete traversal of the input data using the Active Key. Vizini v2 performs two passes in opposite directions.
- Directional Agnosticism : The property that the transformation process is symmetrical. Applying the transformation with inverted parameters (pass order and S-box direction) restores the original data.