NSIGII RIFT V1 - Zero Trust Service Architecture

OBINexus Computing Framework

Version: 1.0-JAN2026
Author: Nnamdi Michael Okpala
License: OBINexus Proprietary

Overview

NSIGII RIFT V1 is a comprehensive compiler toolchain implementing:

- **Zero Trust Architecture** via Phantom ID encoding
- **Color-based Verification** (RGB consensus model)
- **CISCO Self-Balancing Trees** (Eulerian bottom-up augmented tries)
- **Multi-stage RIFT Compilation** (Stages 000-333)
- **Tomographic Index Verification** (6-permutation coherence)
- **Service Schema:** `obinexus.[operation].[service]`

Core Philosophy

> **All squares are rectangles, not all rectangles are squares.**
> **All drivers are binders, not all drivers execute logic.**
> **Implicit RIFT means explicit RIFT policies.**

Architecture

Color Verification Layers

Color	Role	Polarity	Share
RED	Incoming data	Positive	1/4
GREEN	Verification	Negative	1/4
BLUE	Outgoing data	Neutral	-
CYAN	RED + GREEN consensus	Neutral	1/2 (1/4 + 1/4)

RIFT Stages

Stage	Name	File Type	Purpose
000-111	Lexer	`.rf`	Token triplet generation
222	Parser	`.mrf`	Canonical tomography AST
333	Semantic	`.mrf`	Unified protocol AST

Token Triplet Structure

```c
typedef struct {
    TokenType type;      // What it is (relation)
    uint32_t memory;     // Where it lives (pointer)
    uint32_t value;      // What it contains (length)
} TokenTriplet;
```

Principle: Memory precedes type, type precedes value.

Directory Structure

```
NSIGII_RIFT_V1JAN2026/
├── c/
│   ├── include/
│   │   └── service/
│   │       ├── nsigii.h
│   │       ├── rift_stages.h
│   │       ├── color_verify.h
│   │       ├── phantom_id.h
│   │       └── cisco_tree.h
│   ├── src/
│   │   ├── nsigii_core.c
│   │   ├── tokenizer.c
│   │   ├── parser.c
│   │   ├── cisco_balance.c
│   │   ├── phantom_encoder.c
│   │   └── color_verify.c
│   └── Makefile
│
├── cpp/
│   ├── include/ (same structure as C)
│   └── src/ (C++ wrappers)
│
├── csharp/
│   ├── NSigii.Rift/
│   │   ├── Context.cs
│   │   ├── Token.cs
│   │   └── NSigii.Rift.csproj
│   └── NSigii.Rift.sln
│
├── python/
│   ├── pyproject.toml
│   ├── setup.py
│   ├── nsigii/
│   │   ├── __init__.py
│   │   ├── rift_binding.py
│   │   ├── phantom_id.py
│   │   └── cisco.py
│   └── tests/
│
├── go/
│   ├── go.mod
│   ├── go.sum
│   ├── nsigii/
│   │   ├── rift.go
│   │   ├── phantom.go
│   │   └── cisco.go
│   └── examples/
│
├── lua/
│   ├── nsigii-rift-1.0-1.rockspec
│   ├── nsigii.lua
│   └── nsigii/
│       ├── rift.lua
│       ├── phantom.lua
│       └── cisco.lua
│
├── riftfiles/
│   ├── *.mrf (metacanonical files)
│   └── *.rf (physical rift stage files)
│
├── docs/
│   ├── API.md
│   ├── ARCHITECTURE.md
│   ├── RIFT_STAGES.md
│   └── ZERO_TRUST.md
│
└── examples/
```

```
    ├── test_input.rift
    ├── tokenize_example.c
    ├── parse_example.py
    └── cisco_balance.go
```

Building from Source

Prerequisites

```bash
# C/C++ toolchain
gcc >= 9.0 or clang >= 10.0
make >= 4.0
cmake >= 3.15

# Python
python >= 3.8
pip >= 21.0

# Go
go >= 1.18

# Lua
lua >= 5.1
luarocks >= 3.0

# C#
dotnet >= 6.0
```

Build Steps

1. Build C Library

```bash
cd c/
make clean
make all

# Output:
#   lib/libnsigii_rift.a
#   lib/libnsigii_rift.so
```

2. Build Python Binding

```bash
cd python/
pip install -e .

# Or with pyproject.toml:
python -m build
pip install dist/nsigii_rift-1.0.0-*.whl
```

3. Build Go Module

```bash
cd go/
go mod tidy
go build ./...
go test ./...
```

4. Build Lua Module

```bash
cd lua/
luarocks make nsigii-rift-1.0-1.rockspec
```

5. Build C# Assembly

```bash
cd csharp/
dotnet build NSigii.Rift.sln
dotnet pack -c Release
```

Usage Examples

C Example

```c
```

```c
#include <nsigii.h>

int main() {
    // Create context
    NSigiiContext* ctx = nsigii_create_context("tokenize", "lexer");

    // Tokenize source
    const char* source = "let x = 42;";
    TokenTriplet tokens[1000];
    size_t count;

    nsigii_tokenize(ctx, source, tokens, 1000, &count);

    // Print schema
    char schema[256];
    nsigii_generate_schema(ctx, schema, 256);
    printf("Schema: %s\n", schema);  // obinexus.tokenize.lexer

    // Cleanup
    nsigii_destroy_context(ctx);
    return 0;
}
```

Python Example

python

```python
from nsigii import NSigiiContext, tokenize

# Method 1: Context manager
with NSigiiContext("tokenize", "lexer") as ctx:
    tokens = ctx.tokenize("let x = 42;")
    print(f"Schema: {ctx.schema}")
    for token in tokens:
        print(token)

# Method 2: Convenience function
tokens = tokenize("let result = (x + y) * 2;")
print(f"Generated {len(tokens)} tokens")
```

Go Example

go

```
package main

import (
    "fmt"
    "log"
    "github.com/obinexus/nsigii-rift/nsigii"
)

func main() {
    // Create context
    ctx, err := nsigii.NewContext("tokenize", "lexer")
    if err != nil {
        log.Fatal(err)
    }
    defer ctx.Close()

    // Tokenize
    tokens, err := ctx.Tokenize("let x = 42;")
    if err != nil {
        log.Fatal(err)
    }

    // Print results
    schema, _ := ctx.Schema()
    fmt.Printf("Schema: %s\n", schema)

    for _, token := range tokens {
        fmt.Println(token)
    }
}
```

Lua Example

```
lua
```

```lua
local nsigii = require("nsigii")

-- Method 1: Context
local ctx = nsigii.new_context("tokenize", "lexer")
local tokens = ctx:tokenize("let x = 42;")
print("Schema: " .. ctx:schema())

for i, token in ipairs(tokens) do
    print(string.format("%d: %s '%s'", i, token.type_name, token.text))
end

ctx:close()

-- Method 2: Convenience function
local tokens = nsigii.tokenize("let result = (x + y) * 2;")
local stats = nsigii.analyze_tokens(tokens)
print("Total tokens: " .. stats.total_tokens)
```

RIFT Files

.rf (Rift Files) - Physical Stage

rift

```
// test_input.rift
// RIFT Test Program - Stage 000-111

let result = (x + y) * 42;
let factorial = n * factorial(n - 1);

function calculate(a, b, c) {
    let intermediate = a + b;
    return intermediate * c;
}
```

.mrf (Meta Rift Files) - Metacanonical

xml

```
<!-- example.mrf - Metacanonical Representation -->
<rift stage="222" tomography="enabled">
  <declaration type="let" id="result">
    <expression op="mul">
      <group op="add">
        <identifier>x</identifier>
        <identifier>y</identifier>
      </group>
      <literal type="number">42</literal>
    </expression>
  </declaration>
</rift>
```

AUX Instruction Sequence

Instruction	Value	Meaning
`AUX_NOSIGNAL`	0x00	Half-start (no signal)
`AUX_SIGNAL`	0x01	Dual-start (signal present)
`AUX_START`	0x02	Full initialization
`AUX_STOP`	0x03	Termination with context

Noise Levels

- **NOISE_HIGH (1)**: High entropy initialization
- **NOISE_LOW (0)**: Low entropy/deterministic start

Principle: AUX instructions are half of the full instruction in sequence for signal processing of imagery bytes.

CISCO Self-Balancing Tree

CISCO = Eulerian Bottom-up Self-Balancing Tree Augmented Trie Node Chain

Properties

- Real-time platform-independent instruction aux start/stop
- RED-GREEN-BLUE verification during tree operations
- Self-balancing on token insertion

- Eulerian path for optimal traversal

Operations

```c
CiscoTree* tree = nsigii_cisco_create();
nsigii_cisco_insert(tree, token, COLOR_RED);
bool balanced = nsigii_cisco_verify_balance(tree);
nsigii_cisco_rebalance(tree);
nsigii_cisco_destroy(tree);
```

Zero Trust: Phantom ID

Generation

```c
PhantomID id;
VerificationKey key;
nsigii_phantom_generate(ctx, &token);
```

Verification

```c
bool valid = nsigii_phantom_verify(&id, &key, &token);
```

Properties

- Salt-based cryptographic identity
- No storage of raw identity data
- Derived IDs for purpose separation
- SHA-512 hash with encoding map

Testing

C Tests

```bash
```

```
cd c/
make test
./tests/run_tests
```

Python Tests

```bash
cd python/
pytest tests/
```

Go Tests

```bash
cd go/
go test -v ./...
```

Lua Tests

```bash
cd lua/
lua tests/test_nsigii.lua
```

Service Schema Examples

```
obinexus.tokenize.lexer      # Stage 000-111: Tokenization
obinexus.parse.syntax        # Stage 222: Parsing
obinexus.analyze.semantic    # Stage 333: Semantic analysis
obinexus.verify.color        # RGB color verification
obinexus.balance.cisco       # CISCO tree balancing
obinexus.encode.phantom      # Phantom ID encoding
```

Performance

Benchmarks (AMD64, 3.0GHz)

Operation	Time	Throughput
Tokenization	~0.5ms	2000 tokens/sec
Parser	~1.2ms	800 AST nodes/sec
CISCO Insert	~0.1ms	10000 ops/sec
Phantom Generate	~0.3ms	3000 ids/sec

Contributing

1. Fork the repository
2. Create feature branch
3. Implement with tests
4. Submit pull request

Code Style

- **C:** Follow Linux kernel style
- **Python:** PEP 8
- **Go:** gofmt
- **Lua:** LuaStyle

License

Contact

- **Author:** Nnamdi Michael Okpala

- **Email:** nnamdi@obinexus.com
- **GitHub:** https://github.com/obinexus/nsigii-rift
- **Documentation:** https://nsigii-rift.readthedocs.io

References

1. Okpala, N.M. (2025). *Zero-Knowledge Proofs: Mathematical Foundation*
2. Okpala, N.M. (2025). *Isomorphic Reduction — Not a Bug, But a Feature*
3. Okpala, N.M. (2026). *NSIGII Protocol Stream Transcript*
4. OBINexus Computing. *RIFT Specification V1*

Structure is the final syntax.