

# Iperbit HTC3-LR v1.1

## Phase 2 – Deterministic Decode

Author: Luca Rossetto  
Independent Researcher

Latina, Italy

---

## 1. Overview

HTC3-LR (Low-RAM) is a deterministic, streaming-capable entropy codec designed for constrained environments.

Phase 2 focuses on:

- Deterministic decode validation
- Container parsing
- Structural corruption detection
- Optional CRC32 integrity verification
- Reproducible performance measurement

The decoder operates without heap allocation and reconstructs canonical Huffman tables from container-embedded packed representations.

---

## 2. Container Structure

An HTC3 container consists of:

- Magic header ("HTC3")
- Version byte
- Seed byte
- Packed canonical tables
- Bitstream section (bsLen)
  - rawLen (uint32 LE)
  - rawCRC (uint32 LE, ver  $\geq 2$  only)
  - compressed payload

### Versioning

- **Version 1:** No CRC present
- **Version 2:** Includes CRC32 (IEEE) of decoded raw data

---

### 3. Deterministic Decode Model

The decoder guarantees:

- Deterministic reconstruction of canonical tables
- Streaming bit consumption
- No dynamic memory allocation
- Explicit failure on malformed input

Decoding is complete only if:

```
decoded_bytes == rawLen
```

Any mismatch results in **FAIL**.

---

### 4. Integrity Model

Integrity validation operates on two levels:

#### 4.1 Structural Validity (All Versions)

The decoder must reconstruct exactly `rawLen` bytes.

Failure conditions include:

- Malformed header
- Premature EOF
- Invalid canonical table reconstruction
- Bitstream decode failure
- Incomplete reconstruction (`decoded < rawLen`)

If any of these occur → **FAIL**

---

#### 4.2 CRC-Verified Integrity (Container v2)

When `ver >= 2`:

- CRC32 (IEEE polynomial 0xEDB88320) is computed over decoded raw data.
- If computed CRC  $\neq$  stored CRC → **FAIL**

For container **v1**, CRC is not present.

In this case:

- Structural validity is enforced.
- CRC verification is reported as **N/A**.
- PASS is returned if structural reconstruction is correct.

This preserves compatibility while allowing stronger guarantees in v2.

---

## 5. Corruption Testing

### 5.1 Payload Bit Flip (Container v1)

Procedure:

- A single byte inside the compressed payload was modified.
- File length unchanged.
- Decode executed normally.

Observed behavior:

```
INPUT EOF before decode finished  
FAIL
```

The decoder detected structural inconsistency due to invalid bitstream progression.

Result:

- No crash
- No undefined behavior
- Deterministic FAIL

This confirms structural corruption detection is active even without CRC.

---

### 5.2 Structural vs Semantic Corruption

Without CRC (v1), the system detects:

- ✓ Invalid Huffman codes
- ✓ Premature EOF
- ✓ Structural inconsistencies
- ✓ Length mismatches

It cannot guarantee detection of:

- ✗ Bit flips that still produce valid symbol sequences
- ✗ Semantic modifications that preserve length

CRC (v2) closes this gap.

---

## 6. Performance Benchmark

### Test Dataset

- JSON sample
- rawLen = 4,131,054 bytes (~3.94 MiB)
- Compressed payload  $\approx$  2,057,473 bytes

### Build Environment

- Windows 10 x64
- MinGW64
- GCC -O3

### Measured Results

```
decode_time: ~1.21 s
decode_speed: ~32.5 MB/s
ratio_payload: 0.4981 (~49.8%)
ratio_container: 0.4981 (~49.8%)
```

Interpretation:

- ~50% compression ratio on structured JSON
- Deterministic streaming decode at ~32 MB/s
- Zero heap allocation

These numbers are reproducible using the included test file.

---

## 7. Memory Model

HTC3-LR:

- Uses fixed-size structures
- No dynamic allocation during decode
- O(1) symbol lookup per decoded symbol
- Canonical codes reconstructed at load time

This design targets:

- Embedded systems
  - Streaming IoT devices
  - Deterministic processing environments
-

## 8. Reproducibility

### Build (Optimized)

```
gcc -O3 example_decode.c htc3_lr.c -o test_htc3
```

### Run

```
./test_htc3.exe test_stream.bin out.raw
```

### JSON Validation (Node.js)

```
node -e "const fs=require('fs'); JSON.parse(fs.readFileSync('out.raw','utf8')); console.log('OK JSON');"
```

---

## 9. Design Principles

HTC3-LR prioritizes:

- Determinism
- Structural integrity
- Low memory footprint
- Canonical table reconstruction
- Explicit failure semantics

It does not attempt to compete with high-throughput LZ-family codecs. Instead, it targets environments where:

- RAM is constrained
  - Deterministic behavior is required
  - Streaming decode is mandatory
- 

## 10. Status

Phase 2 – Deterministic Decode:

- ✓ Canonical reconstruction verified
- ✓ Structural corruption detection verified
- ✓ CRC-based integrity (v2) supported
- ✓ Performance measured and reproducible
- ✓ Streaming behavior validated

The decoder is stable and suitable as a reference implementation for HTC3-LR.