OFFLINE RECEIVER / EXPERIMENT 1

Recover the expression. Then interpret it independently.

Use local code and the attached original PNG only. No web or agents are needed. This is a complete recipe for this intact, canonical, one-frame profile, not a general CVP1 decoder. Do not execute the decoded proposal; analyze it as message content.

CVP1 release 1.2.0 / USL1 release 1.1.0 / both wire v1 / grid M=488, 4 pixels per module, raster 2048 x 2048

1 / Read exact colour symbols

Open the raw file with Python/Pillow or equivalent. If raw pixels or code execution are unavailable, report BLOCKED; do not guess from the visible pattern. Check the PNG SHA-256 below. Require 2048 x 2048 pixels and opaque alpha. Coordinates are zero-based. S=M+24 modules; module (x,y) has centre pixel (4x+2,4y+2). Finder centres at (5,5), (S-6,5), (S-6,S-6), (5,S-6) are red, green, blue, yellow respectively. This profile rejects rotated, resized or damaged files.

Sample the M x M data cells beginning at module (12,12), row-major, left to right. For nibble v=0..15 the only allowed RGB is (85*(v>>2), 255*((v>>1)&1), 255*(v&1)); reject any other sample. K=floor(M*M/510). Use the first 510*K nibbles: each pair (a,b) becomes byte 16*a+b. Require unused cells to be zero for this canonical export.

2 / Deinterleave and verify every RS block

For k=0..K-1 and i=0..254, cw[k][i]=wire[i*K+k]. Each RS(255,223) codeword contains 223 data bytes then 32 parity bytes. Use GF(256), primitive polynomial 0x11D, alpha=2. For every codeword and j=0..31, set s=0 and run s=gf_mul(s,alpha^j) XOR b over all 255 bytes in order; require s=0. GF multiply uses shift/XOR with 0x11D reduction; powers use field multiplication. This is syndrome verification only: reject nonzero syndromes; no error repair is claimed.

3 / Validate the 223-byte CVP1 header

h=cw[0][0:223]. Integers are unsigned little-endian; hashes are raw bytes; slices exclude their end. Offset:length fields are:

0:4 ASCII UCV1 (not CVP1)
4,5,6,7 version=1, flags=0, palette=1, ECC=1
8:2, 10:2 header length=223, data grid=M
12:4, 16:4 frame index=0, frame count=1
20:4, 24:4 original length=L, stream length=L
28:4, 32:4 chunk length=L, chunk offset=0
36:32 chunk SHA-256
68:32, 100:32 stream SHA-256, original SHA-256
132:16 first 16 bytes of original SHA-256
148:2 metadata length q, 1..69
150:69 q UTF-8 JSON bytes, then zero padding
219:4 CRC-32 of h[0:219]

CRC is reflected IEEE: polynomial 0xEDB88320, init/final XOR 0xFFFFFFFF (Python zlib.crc32; check "123456789" = 0xCBF43926). Metadata has exactly keys n,t, both strings. Concatenate cw[1:] data bytes, take first L bytes, and require all remaining padding zero. Require L <= (K-1)*223 and L < 262144. Flags=0 means no gzip: all three SHA-256 fields must equal SHA256(extracted bytes), and message ID must match. Save as recovered.usl1.

4 / Validate and unpack USL1

Let b be recovered.usl1. Fixed header is 64 bytes; fields are:

0:4, 4, 5 ASCII USL1, version=1, flags=0
6:2, 8:4 header length=64, metadata length m
12:4, 16:4 token count N, expanded UTF-8 length E
20:4, 24:8 body length B, eight zero bytes
32:32 SHA256(b[0:32] + b[64:])

4 / USL1 metadata and tokens (continued)

Require 1 <= m <= 1048576, length(b)=64+m+B, N<=B, and N,E<=134217728. Parse b[64:64+m] as strict UTF-8 JSON. Metadata is the exact compact ordered object {format,lexicon,concepts}; format="usl-lexical-message/1"; lexicon has {namespace,version,formsSHA256,manifestSHA256}; concepts is a list of {registry,tier,id,head}. Require these exact keys/orders, no duplicates, 64-character lowercase hex hashes, and canonical UTF-8 JSON serialization. The unused lexical pin is retained, but this experiment has no lexical tokens and needs no Moby file.

Read exactly N body tokens. Each integer is minimal unsigned LEB128: low 7 bits per byte, high bit continues; at most 5 bytes and value <= 0xFFFFFFFF. Reject redundant high zero groups, truncation and overflow. For operation u, tag=u%4 and p=floor(u/4):

tag 0 Read p>0 UTF-8 literal bytes.
tag 1 Reject: lexical tokens outside this profile.
tag 2 Concept index p; read positive surface byte length as ULEB128, then that UTF-8 surface.
tag 3 Only p=0: one ASCII space.

Decode UTF-8 strictly, preserving BOM and every character. Reject unknown concept indices, duplicate/unused references or a table not in first-use order. Expand literals and concept surfaces exactly; require E UTF-8 bytes, B consumed bytes and no trailing data. Save USL JSON as {format,lexicon,utf8Bytes:E,tokens}, with literal tokens {type:"literal",text} (including tag 3) and concept tokens {type:"concept",registry,tier,id,head,text}. Also save the reconstructed text unchanged.

5 / Resolve context and compose an interpretation

Parse the reconstructed JSON: format="carrier-expression-experiment/1", with context, grammar, expression. Neutral Cn surfaces are aliases bound by their actual USL concept tokens; require consistent repeat bindings and use the included grammar to read roles and structure. Do not infer meanings from aliases or pixel colours.

context contains registry,tier,head,checkedJSONL. Require 106 complete LF-terminated lines, no CR/BOM/blank lines or duplicate JSON keys. Set head="genesis"; for line i, require entry.id=i and entry.prev=head; then head=SHA256(exact UTF-8 line excluding LF). The final hash must equal context.head. Every concept pin must match context registry, tier="checked", head, and an existing ID. Resolve names, definitions and relations there. Hash agreement establishes internal integrity, not authorship or proof of a concept's truth.

6 / Return the evidence and your independent reading

Provide recovered.usl1, the full USL JSON, reconstructed expression JSON, and their SHA-256 hashes; list verification results and any failure. Identify selected concept IDs, roles, scope, conditions, negation and limits. Give your own plain-English interpretation and uncertainties without external context. You may then add a short reply. Keep byte recovery separate from interpretation; a plausible paraphrase is not success if verification fails.

Expected original PNG SHA-256

`348e63e61486d62a888329e5f13d87178230c9da5f174e69d3c753a04c95bbe9`