Oboron Unauthenticated Layer (obu) Specification

oboron.org

version 1.0

The **obu** layer specifies Oboron's *unauthenticated* schemes. It is a layered extension of the Oboron Protocol Specification: the format model and encodings are defined there and are not repeated here. This layer is performance-oriented and uses its own secret (§3); it is normative for the schemes it defines, and also specifies the optional `obu` command-line interface (§6).

Security notice. The schemes in this layer do *not* provide integrity protection and MUST NOT be used where authentication is required; use the authenticated Oboron core for any security-critical application. The detailed requirements are in §4.

> `upcbc` provides confidentiality without authentication and is vulnerable to ciphertext tampering; `zdcbc` is obfuscation only and is *not* cryptographically secure.

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in BCP 14 (RFC 2119, RFC 8174) when, and only when, they appear in all capitals, as shown here.

1 Schemes

This layer defines two schemes. A scheme ID has three parts: a tier letter, a property letter, and an algorithm code — e.g. `upcbc` is `u` + `p` + `cbc`. (Core scheme IDs have two parts, property + algorithm; the obu tier letter prefixes that to mark the security gradient.)

Tier (first letter)

- **`u` — Unauthenticated.** Confidentiality only, no integrity protection. Example `upcbc`. Suitable only when integrity is unnecessary, or is provided by a separate cryptographic mechanism. *Warning*: vulnerable to ciphertext tampering.
- **`z` — Obfuscation.** *Not* cryptographically secure — for non-security use only. Example `zdcbc`: deterministic obfuscation with a constant IV. OPTIONAL (§5).

Property (second letter)

- **`p` — probabilistic.** Different output each time (random IV). Example `upcbc`.
- **`d` — deterministic.** The same plaintext always produces the same obtext, usable as a stable, referenceable handle. Example `zdcbc`.

Algorithm (remaining letters)

Both schemes use `cbc` = AES-CBC.

Summary

Scheme	Algorithm	Deterministic
`upcbc`	AES-256-CBC	No
`zdcbc`	AES-128-CBC	Yes

`upcbc` MUST use 256-bit AES; `zdcbc` MUST use 128-bit AES.

2 Algorithm

As in the core, an obtext is the encoding of the scheme's ciphertext output; the scheme is supplied by the caller and is not carried in the obtext.

2.1 Padding

The CBC schemes pad the plaintext to a multiple of the AES block size (16 bytes) with `0x01` bytes. When the plaintext length is already a multiple of 16, no padding is added — this keeps short obtexts compact. On decryption, trailing `0x01` bytes are stripped.

`0x01` is U+0001 (SOH), a control character that does not occur at the end of normal text. To keep the padding unambiguous, `enc` MUST reject any plaintext whose final byte is `0x01`; with that guardrail, stripping trailing `0x01` bytes on decryption always recovers the exact plaintext.

Because `upcbc` is unauthenticated, implementations SHOULD report every `dec` failure — structural (§2.2) and plaintext-validity (UTF-8) alike — through a single uniform error, so that `dec` does not become a distinguishing oracle. The padding step itself never fails: stripping trailing `0x01` always succeeds.

2.2 Per-Scheme Ciphertext Layout

Implementations MUST conform to these layouts for cross-implementation interoperability.

- **`upcbc` (AES-256-CBC):** draws a fresh 16-byte IV from a cryptographically secure random number generator before each encryption; if suitable randomness is unavailable, `enc` MUST fail. Layout: `16-byte random IV || padded ciphertext`.
- **`zdcbc` (AES-128-CBC):** uses a *constant* IV (§3), so encryption is deterministic. After CBC encryption it applies the prefix restructuring of §2.3. The IV is fixed and is *not* included in the output. Layout: `restructured padded ciphertext`.

Decryption input validation. After decoding, `dec` MUST reject any scheme output that cannot match the scheme's layout: for `upcbc`, an input shorter than 32 bytes (a 16-byte IV

plus at least one 16-byte block) or whose post-IV length is not a positive multiple of 16; for `zdcbc`, an input whose length is not a positive multiple of 16. As in the core, the empty plaintext is outside the obu plaintext domain: `enc` MUST reject it, and `dec` MUST reject any input that decrypts and strips to zero bytes.

2.3 zdcbc Prefix Restructuring

`zdcbc` applies an additional byte-wise XOR transformation to the CBC ciphertext, mixing the final block into the first so that more of the plaintext's variation surfaces in the output prefix:

```
if len(ciphertext) > 16:        // more than one AES block
    ciphertext[0..16] ^= ciphertext[len-16 .. len]
```

Byte ranges are half-open: `[0..16]` is the first 16-byte block and `[len-16 .. len]` is the last. The transform is its own inverse and MUST be reversed during decryption, before the standard CBC decryption. A single-block ciphertext (`len` = 16) is left unchanged.

3 Key Material

The obu layer uses a single 256-bit **obu secret** (canonically the `OBORON_SECRET` environment variable), separate from the core's master key. Both schemes draw their key material from it directly; no key derivation is performed:

- **upcbc:** the full 32-byte secret is the AES-256-CBC key. The IV is drawn fresh per encryption (§2.2).
- **zdcbc:** the AES-128-CBC key is bytes `0..16` of the secret; the constant IV is bytes `16..32` (half-open ranges, as in §2.3).

The secret is 32 bytes, encoded as 64 lowercase hexadecimal characters; implementations MUST also accept 32 raw secret bytes, and MUST reject a secret of any other length. A generated secret MUST be drawn from a cryptographically secure random number generator. Independence of the obu secret from the core master key is required by §4.

4 Security Considerations

These schemes are deliberately weak. This section states what they do and do not protect, and the requirements that keep them from being misused.

No integrity. `upcbc` and `zdcbc` provide no message authentication. `upcbc` ciphertext is malleable: editing it lets an attacker flip chosen plaintext bits (corrupting the following block) undetectably. obu output MUST NOT be used as an authentication tag, MAC, signature, bearer or capability token, integrity check, or as the sole protection for any value whose unauthorized modification must be detected. Where authentication is required, use the authenticated Oboron core.

Key independence. The obu secret MUST be independent key material: it MUST NOT be derived from, or shared with, the Oboron core master key, and SHOULD NOT be reused

with other protocols or libraries. The published test vectors reuse core-key bytes as the obu secret for conformance convenience only; that derivation MUST NOT be imitated in production.

IV unpredictability. `upcbc` confidentiality depends on an unpredictable IV; it MUST come from a cryptographically secure random number generator (§2.2). A predictable or reused IV reintroduces chosen-plaintext distinguishers.

Deterministic leakage. `zdcbc` uses a constant IV, so it is deterministic: equal plaintexts produce equal obtexts, revealing equality and, in low-cardinality domains, frequency. It is reversible obfuscation for anyone holding the secret, not encryption; do not use it where equality or frequency must not leak.

Uniform errors. As in §2.1, `dec` SHOULD report all failures through one uniform error so it does not become a distinguishing oracle.

5 Conformance

The obu layer is OPTIONAL for a conforming Oboron implementation. An implementation that provides it MUST implement `upcbc` and MUST support every `upcbc`–encoding format; `zdcbc` is OPTIONAL, and a build without it MUST reject `zdcbc` formats as unknown. The encodings, format syntax, UTF-8 handling, and non-empty plaintext domain are inherited from the Oboron Protocol Specification.

Conformance is checked against the obu test vectors published at `https://gitlab.com/oboron/oboron-test-vectors` (`obu-test-vectors.jsonl` and its negative companion); an implementation MUST pass the vector set carried by the release tag matching this specification's version. For `zdcbc` (deterministic), implementations MUST produce identical obtexts for the same secret, format, and plaintext; for `upcbc` (probabilistic), they MUST decrypt obtexts produced by any other conforming implementation.

6 Command-Line Interface

Conforming implementations MAY provide an `obu` binary — the optional command-line interface for this layer's schemes; it is OPTIONAL. It is a layered extension of the Oboron CLI Specification: the command-syntax notation, format resolution, secret-resolution precedence, exit codes, and standard I/O behavior (including the `--` end-of-options rule and the byte-exact stdin/stdout framing) are defined there and are not repeated here. If provided, `obu` MUST implement `enc`, `dec`, and `secretgen`, plus the global `--help` and `--version` options. It mirrors the core `ob` CLI, differing only as summarized here:

Aspect	`ob` (core)	`obu`
Key terminology	"key" (512-bit)	"secret" (256-bit)
Key flag	`--key` / `-k`	`--secret`
Environment variable	`OBORON_KEY`	`OBORON_SECRET`
Default scheme	`dsiv`	`upcbc`
Schemes	`dsiv`, …	`upcbc`, `zdcbc`
Key generation	`keygen`	`secretgen`

> `--secret` has no single-letter alias, to avoid confusion with the core's `-s` (`--dsiv`) scheme flag.

The `obu --version` line follows the core CLI's `--version` format with `obu` as the program name.

6.1 Secret

`obu` uses a single 256-bit secret in place of the core's 512-bit key, supplied via `--secret <SECRET>`, the `OBORON_SECRET` environment variable, or `--keyless` (the fixed public test secret, §6.4), resolved with the same precedence and mutual-exclusion rules as the core CLI's key sources. A secret supplied via `--secret` or `OBORON_SECRET` MUST be exactly 64 lowercase ASCII hexadecimal characters; an empty `OBORON_SECRET` is an invalid secret, not an absent one.

6.2 enc / dec

`obu enc` and `obu dec` take the same options as the core `ob enc` / `ob dec`, with `--secret` in place of `--key` and the obu scheme flags below. As in the core, the scheme is supplied by the caller — a scheme flag, a `--format` string, or the built-in default `upcbc` — and is *not* auto-detected from the obtext. `zdcbc` is OPTIONAL (§5); a build without it MUST reject `zdcbc` formats as unknown.

Flag	Short	Description
`--upcbc`	`-u`	Probabilistic AES-256-CBC (unauthenticated).
`--zdcbc`	`-z`	Deterministic AES-128-CBC (obfuscation).

6.3 secretgen

`obu secretgen` prints a freshly generated random 256-bit secret to stdout and exits — the obu counterpart of `ob keygen`. It needs no secret, configuration, or stdin, and creates or modifies no file. The output is 64 lowercase hexadecimal characters followed by a single newline; exit `0`. If a cryptographically secure random number generator is unavailable, `secretgen` MUST fail with status `1` and MUST NOT write a secret.

6.4 Fixed Public Test Secret

For cross-implementation testing, `obu` supports a **fixed public test secret** via the `-K`/`--keyless` flag. Implementations MUST use this exact value for test-vector compatibility. The secret is public and provides no security; implementations SHOULD warn when it is supplied through `--secret` or `OBORON_SECRET` rather than `--keyless`, and MUST suppress that warning on a failed `dec` so that only the uniform decryption-failure message appears. This value is the first 32 bytes of the core CLI's fixed public test key; §4 forbids deriving a production secret this way.

Hex (64 chars) — canonical:

`381284633d02ea5f35df8596b5cc4218310060468e8b465455a415174ea6e966`

Test vectors use the Oboron CLI Specification's JSON Lines format — positive `{format, plaintext, obtext}` and negative `{op, format, input, reason}` — and bind to this secret with no per-vector field. They ship as `obu-test-vectors.jsonl` and `obu-negative-test-vectors.jsonl` (§5), run via `obu`; a negative vector MUST fail with status `1`. The deterministic exact-match and probabilistic decrypt-the-stored-obtext requirements are those of §5.

References

- BCP 14 — RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*; RFC 8174, *Ambiguity of Uppercase vs. Lowercase in RFC 2119 Key Words*.
- FIPS 197 — *Advanced Encryption Standard (AES)*.
- NIST SP 800-38A — *Recommendation for Block Cipher Modes of Operation: Methods and Techniques* (the CBC mode).
- Oboron Protocol Specification — `https://oboron.org/protocol-spec/oboron-v1.0.html` (format model, encodings, and plaintext domain).

Index