Introduction to Data Obfuscation Method (DOM)

Cryptography Basics of DOM

By: Eric Dumouchelle

September 2025

Data Alchemy Security

Abstract

Data Obfuscation Method (DOM) is a new approach to data protection that resists harvest now, decrypt later (HNDL) attacks. Instead of encrypting data into a single ciphertext that can be stockpiled and attacked later, DOM separates information into two inert artifacts:

- A Noise file (N) that is indistinguishable from randomness.
- A Key file (K) that contains only mapping instructions.

Neither artifact reveals anything on its own. Only when the correct pair is combined does the original message reappear. When the noise length n ≥ message length m and noise is never reused, DOM achieves information-theoretic secrecy in the Shannon sense[1]. This paper establishes the basic cryptographic definitions, encoding and reconstruction procedures, and proofs of correctness and secrecy. iDOM, the software implementation of DOM, operationalizes these concepts with TRNG-backed entropy, hybrid mixing, and optional secret sharing.

Background

Encryption encodes data into ciphertext that depends on a secret key. While strong today, such ciphertext remains vulnerable to future breakthroughs in mathematics or computing. This creates the HNDL problem: adversaries can harvest encrypted data now and decrypt it later.

DOM addresses this by removing the concept of a vulnerable ciphertext altogether. Instead, the data is reconstructed through randomness, with only an index (the “key” file) showing how to reconstruct it. Without both artifacts, an attacker learns nothing. This shift from “encrypted file” to “two inert halves” creates an additional line of defense that is independent of algorithmic strength or computational assumptions.

Terminology

For clarity, throughout this paper, the naming convention used in iDOM software is present:

- “Noise file” (.noise)
- “Key file” (.key)

However, in cryptographic terms, the noise file functions as the secret key, and the key file (or index) functions as the ciphertext. Thus, the file names are inverted relative to conventional cryptographic roles.

To avoid misleading practitioners, implementation terms from iDOM are used, but please note that they map inversely to canonical cryptographic roles.

iDOM Term	**Cryptographic Role**
Noise file (.noise)	Secret Key (uniform randomness)
Key file (.key)	Ciphertext (encoding dependent on key)
Plaintext (M)	Message
Reconstruction	Decryption

BLAKE2b: cryptographic hash function used in iDOM's entropy pool to mix and compress inputs (TRNG samples, CSPRNG, and domain IDs) into uniform, unpredictable output[2]

CSPRNG (Cryptographically Secure Pseudo-Random Number Generator): used as part of the entropy pool generation

HNDL (Harvest Now, Decrypt Later): adversary model in which encrypted payloads are harvested today and stored for later decryption once cryptanalytic or computational advances become available

Perfect Secrecy: Condition where ciphertext leaks zero information about the plaintext

SSS (Shamir's Secret Sharing): cryptographic method that divides a secret into multiple parts, called shares, and distributes them among a number of participants[3]

TRNG (True Random Number Generator): non-deterministic entropy source; iDOM implements a TRNG-backed entropy pool during noise and key creation

Adversary Model: Harvest Now, Decrypt Later

The central threat DOM addresses is harvest now, decrypt later (HNDL) attacks. Encryption encodes the plaintext M in a way that is reversible once the key is recovered or the algorithm is broken. Even if strong today, such ciphertext can be decrypted later, even without having the accompanying key.

DOM produces two separate files:

- Noise file (N): pure random bytes generated from TRNG
- Key file (K): mapping instructions that, when applied to N, yield M

Harvesting one component is useless:

- Harvesting N: indistinguishable from uniform randomness
- Harvesting K: meaningless offsets without N
- Only (N, K) together reconstruct M

Unlike encryption, where stealing the ciphertext is always valuable, in DOM, stealing a single artifact is useless. Noise looks like random garbage, and keys look like meaningless offsets; only both together reconstruct the hidden plaintext.

Formalizing Encoding

Let plaintext be M of length m bytes. Let noise be $N \in \{0,1\}^n$ ($n = n\_rows * row\_bytes$ bytes) with $n \geq m$ (for perfect secrecy). Let index file be K, a set of instructions mapping positions in N to recover M.

Encoding:

1. Generate $N \sim U(\{0,1\}^n)$ using TRNG
2. Construct K such that applying K to N yields M

Decoding:

1. Apply K to N to reconstruct M

(intentionally left blank formatting purposes; please continue to next page)

Encoding Pseudocode:

```
def build_key(input, noise, params):
    backend = choose_backend(...)
    backend.build(noise, row_bytes=params.row_bytes, k=max(params.chain))
    backend.open()

    writer = KeyWriter(out_path, params, noise_sha256=sha256(noise))
    i = 0
    while i < len(input):
        for k in (3,2,1):
            chunk = input[i:i+k]
            spot = backend.random_one(chunk)
            if spot:
                row, pos = spot
                writer.write_entry(row, pos, k)
                i += k
                break
    writer.finalize()
```

Proof of Correctness

Goal: Show that for any input file M and generated noise file N, the encoding procedure produces a key file K such that the reconstruction procedure combine(N, K) outputs exactly M, assuming the integrity checks pass and the placement-entropy guardrail holds.

Preconditions and Notation

- M: input byte string of length m
- N: noise matrix of size n = n_rows * row_bytes bytes generated from the TRNG-backed entropy pool
- chain = (3,2,1): chunk sizes, including fallbacks, used by the encoder
- backend: BucketScan or PackedIndex, which indexes N and supports random_one(chunk) that returns a uniformly sampled coordinate (row, pos) among all matches of chunk within N, or None if no match exists this sampling prevents deterministic reuse of positions, ensuring that duplicate plaintext chunks may map to different positions in N.
- Placement-entropy guardrail: except with negligible probability and in cases of small noise files, every 1-byte value appears in N at least once, and the expected density of 2- and 3-byte matches supports progress

Encoding Invariants

1. Progress Invariant: In the main loop

   ```
   while i < m:
       for k in (3,2,1):
           chunk = M[i:i+k]
           spot = backend.random_one(chunk)
           if spot:
               writer.write_entry(row, pos, k)
               i += k
               break
   ```

   whenever spot is found, the index i advances by k >= 1. Because 1 is in chain, if there exists at least one match at k=1, the loop advances on every iteration until i = m.

2. Coverage Invariant: The sequence of written entries (row, pos, k) forms a disjoint tiling of [0, m) by contiguous chunks M[i:i+k] in left-to-right order; i.e., no gaps and no overlaps are possible by construction because i only moves forward and each step copies exactly the next k bytes of M.

3. Match Correctness Invariant: For each written entry (r, p, k), the backend guarantees that N[r, p : p+k] == M[i : i+k] at the time of writing. This holds by the definition of a match returned by random_one.

Key Serialization and Binding

Each entry is serialized as bit-packed triples (row_id, pos, k). The key header includes noise_sha256 and geometry (n_rows, row_bytes).

Reconstruction Procedure

Pseudocode:

```
gen = (noise.read_bytes(r,p,k) for (r,p,k) in key.entries())
rebuild_output_file_from_iter(gen, out)
```

- Geometry/Hash Binding. Before reconstruction, verify(noise, key.noise_sha256) and geometry checks ensure the provided N is exactly the noise instance used during encoding. If the noise differs, reconstruction aborts.
- Chunk Reproduction. For each serialized entry (r,p,k), the NoiseReader yields exactly the slice N[r, p : p+k]. By the Match Correctness Invariant, this equals the original chunk M[i : i+k] recorded at encode time.

- Concatenation. By the Coverage Invariant, concatenating all yielded slices in order reproduces a copy of M. The decoder streams these slices, producing the final output M_hat.

Proof Sketch (Correctness)
Under the guardrail precondition and successful integrity checks, the combine procedure outputs M_hat = M.

Proof Sketch: By Progress, the encoder terminates after emitting a finite sequence of entries that tile [0,m) (Coverage). Each entry's slice in N equals the corresponding chunk of M (Match Correctness). The decoder reads those exact slices from the same N (enforced by geometry and noise_sha256) and concatenates them in order, yielding M byte-for-byte. Therefore M_hat = M.

A full formal proof is left for future work; a proof sketch is provided sufficient to establish invariants.

Termination and Failure Modes

- Termination: Because 1 is in chain, the encoder advances by at least one byte per iteration; it terminates in at most m iterations.
- Guardrail Violation: If the backend reports no match at all levels k in (3,2,1) for some position, the run aborts. Correctness is preserved by refusing to produce a partial key.

Figure 1. Entropy in Encoding: Index and Key Writer

TRNG-backed entropy feeds the index, sampler, and ultimately the key writer (coordinates, not content).

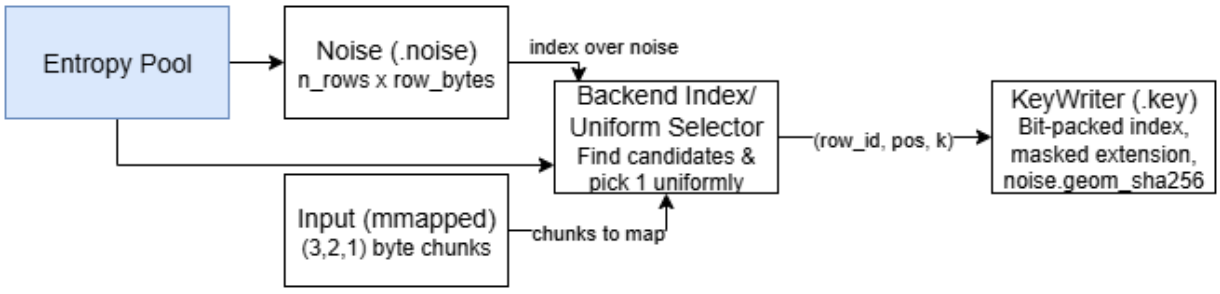


Figure 2. Combine Inputs & Integrity Checks

Inputs are normalized to NoiseReader/KeyReader. Integrity checks must pass before reconstruction proceeds.

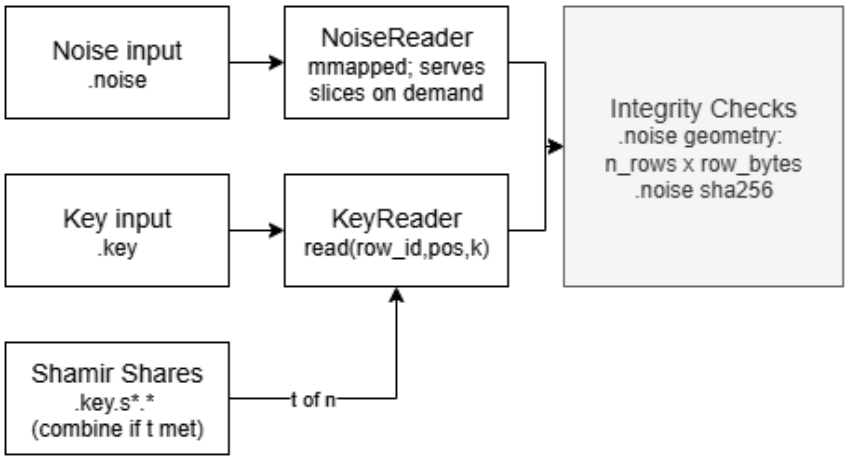
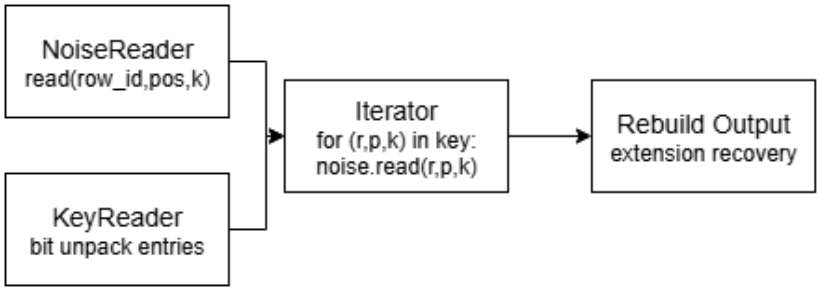


Figure 3. Reconstruction Pipeline

Entries are streamed; extension is restored.



Proof of Secrecy

Case 1: Noise alone

- N is uniformly random
- I(M;N) = 0. No information about M can be inferred

Case 2: Key alone

- K is pointers/offsets without reference
- For any two plaintexts M_1, M_2, distributions of K are indistinguishable without N

- $I(M;K) = 0$
- Equivalently: given only K, the distribution over possible plaintexts remains uniform, conditioned on the absence of N

Case 3: Noise + insufficient key shares

- With SSS, fewer than threshold shares leak no information
- Noise without sufficient shares cannot reconstruct M

Case 4: Sufficient key shares alone

- Sufficient shares without the noise file cannot reconstruct M

Perfect Secrecy Condition

DOM satisfies Shannon-style perfect secrecy when $n \geq m$ and N is never reused. When $n < m$, DOM retains resistance to HNDL by rendering harvested artifacts individually information-free. Strength against HNDL:

- Resistant: if $n < m$
- Immune: if $n \geq m$ (and N never reused)

Bridge to iDOM

The theoretical proofs above are directly implemented in iDOM:

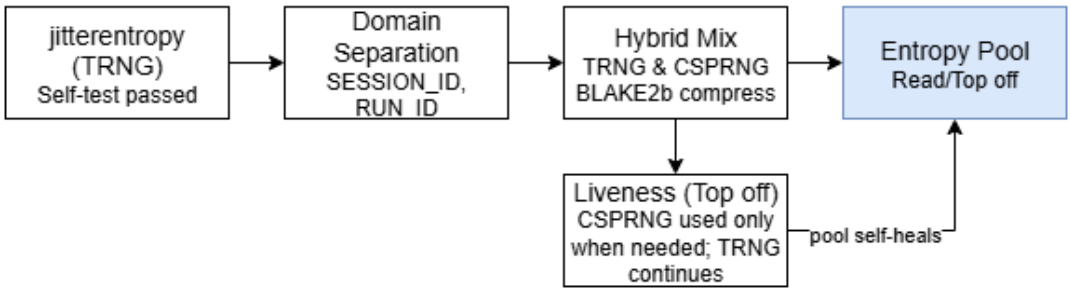
- **TRNG-backed entropy pool:** ensures N is fresh and unique
- **Hybrid entropy mixing:** TRNG + CSPRNG with BLAKE2b domain separation
- **Noise creation:** created during encoding; enforce $n \geq m$ when required
- **Super Shamir's Secret Sharing:** splits K into shares while treating N as a super-share

Entropy Pseudocode:

```
def random_bytes(n):
    if TRNG_enabled:
        return pool.read_or_topoff_with_os_urandom(n)
    else:
        return os.urandom(n)
```

Figure 4. Entropy Pool (TRNG-backed, hybrid hardened)

TRNG samples are domain separated and BLAKE2b-compressed with OS CSPRNG. Pool supports top off for liveness; TRNG remains part of every enabled run.



Security Properties Summary

- **Noise-only:** indistinguishable from randomness; leaks nothing
- **Key-only:** coordinate mappings; inert without noise
- **Both required:** legitimate reconstruction of M
- **Condition for immunity:** n ≥ m, unique N, no reuse

DOM therefore provides information-theoretic secrecy under specified conditions and resistance otherwise, addressing the HNDL adversary model in a way fundamentally different from encryption.

Conclusion

This whitepaper has established that DOM and its implementation in iDOM:

1. Correctly reconstruct the plaintext when both artifacts are available
2. Provide provable secrecy when artifacts are harvested individually
3. Achieve immunity to HNDL attacks under the perfect secrecy condition (n ≥ m, fresh noise)

iDOM operationalizes these principles with TRNG-backed entropy, integrity checks, and optional governance control. By separating payload into two inert artifacts, DOM provides a provable defense against HNDL threats, offering a new level of data security. Future white papers will explore performance, comparison to known systems, compliance, and enterprise integration considerations, but this initial paper establishes the cryptographic foundation.

References

1. Shannon, C.E. Communication Theory of Secrecy Systems. *Bell Syst. Tech. J.* **1949**, *28* (4), 656–715.
2. Aumasson, J.-P.; Neves, S.; Wilcox-O'Hearn, Z.; Winnerlein, C. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). *RFC 7693* **2015**, 1-43.
3. Shamir, A. How to Share a Secret. *Commun. ACM* **1979**, *22* (11), 612–613.