

Cipher Maps: Total Functions as Trapdoor Approximations

Alexander Towell
lex@metafunctor.com

March 2026

Abstract

Privacy in encrypted computation need not come from access-pattern hiding (ORAM), exact algebraic homomorphism (FHE), or simulation-based security (garbled circuits). We show that a *cipher map*—a total function on bit strings where the concepts of domain, correctness, and function identity exist only behind a one-way trapdoor—provides a distinct, quantifiable privacy model. The entire construction reduces to a single design choice: an *acceptance predicate* that partitions hash space among output values. Shannon-optimal allocation of this partition simultaneously minimizes space (achieving the information-theoretic lower bound of $-\log_2 \varepsilon + H(Y)$ bits per element), maximizes output indistinguishability (noise and real outputs share the same frequency profile), and enables predictable error composition ($\eta_{\text{total}} \leq 1 - \prod(1 - \eta_i)$). We formalize the framework through four measurable properties, prove the composition and space optimality theorems, and demonstrate the construction on arbitrary maps, set membership, and encrypted search.

1 Introduction

Consider a setting in which a trusted party wishes to outsource computation to an untrusted machine. The trusted party holds a function $f : X \rightarrow Y$ —a lookup table, a set membership predicate, an index—and wants the untrusted machine to evaluate f on encoded inputs without learning f , X , or Y .

The central object of this paper is the *cipher map*: a total function $\hat{f} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ that the untrusted machine evaluates blindly. The key insight is that \hat{f} is total—every n -bit input produces an n -bit output, with no concept of “in-domain” or “out-of-domain.” The notions of domain, correctness, false positive, and false negative exist only on the trusted machine, which holds the encoding and decoding functions (the trapdoor). The untrusted machine sees only opaque bit strings flowing through opaque total functions.

This paper develops the theory of cipher maps in a self-contained way. We define four properties that characterize a cipher map (§4), formalize the trusted/untrusted machine model (§5), develop the batch construction unified under the acceptance predicate framework (§6), prove the composition theorem governing error accumulation (§7), and discuss the online alternative via trapdoor Boolean algebra (§9.3).

Bernoulli error model. The error framework underlying cipher maps is the Bernoulli model [20]. Two axioms (element-wise independence and conditional independence of block error rates) reduce the error model from exponential complexity to two parameters per element. This tractability is what makes the composition theorem (Property 4) possible: errors combine predictably because they are independent across elements. The Bernoulli model is developed independently; we reference it here only for the error accounting that cipher maps inherit.

What this is not. Cipher maps are not oblivious RAM [13]: ORAM reshuffles storage to hide access patterns with polylogarithmic overhead, while cipher maps use static total functions with noise injection. Cipher maps are not fully homomorphic encryption [12]: FHE preserves exact algebraic structure on ciphertexts, while cipher maps are approximate and hash-based. The closest relative is garbled circuits [22]—both use encrypted lookup tables—but garbled circuits are one-time use and exact, while cipher maps are reusable and approximate with noise closure.

2 Related Work

Cipher maps draw on and differ from several lines of work in data structures, cryptography, and secure computation.

Membership data structures. Bloom filters [6] are the canonical approximate membership structure: a bit array with k hash functions, trading false positives for space. Quotient filters [5] and cuckoo filters [10] improve cache locality and support deletion, respectively, but share the same basic model. Set membership (Remark 6.2) subsumes Bloom filters as a special case: a single hash function ($k = 1$), exact correctness ($\eta = 0$), and the same information-theoretic space bound of $-\log_2 \varepsilon$ bits per element. The difference is that the HashSet is a cipher map—a total function on bit strings with a trapdoor—while Bloom filters expose their output directly.

Perfect hash functions. Fredman, Komlós, and Szemerédi [11] introduced FKS perfect hashing with $O(n)$ space and $O(1)$ lookup. Belazzougui, Botelho, and Dietzfelbinger [2] achieved minimal perfect hash functions near the information-theoretic lower bound. The batch cipher map construction (§6) is a perfect hash function with a different optimization target: it maps domain elements to prefix-free codewords (not to consecutive integers) and trades correctness ($\eta > 0$) for space, achieving $(1 - \eta)(-\log_2 \varepsilon + H(Y))$ bits per element.

Property-preserving encryption. Order-preserving encryption (OPE) [1, 7] and deterministic encryption [4] enable computation on ciphertexts by preserving algebraic structure (order, equality). Naveed et al. [16] demonstrated devastating inference attacks exploiting exactly this preserved structure. Cipher maps differ fundamentally: rather than preserving structure at the cost of leakage, cipher maps hide structure behind a total function. The untrusted machine cannot distinguish domain elements from noise, and the output distribution is controlled by the representation uniformity parameter δ .

Searchable symmetric encryption. Song, Wagner, and Perrig [19] introduced practical encrypted keyword search. Curtmola et al. [9] formalized SSE with simulation-based security (IND-CKA). Cash et al. [8] extended SSE to Boolean queries with sublinear search. Islam et al. [14] showed that access-pattern leakage in SSE enables query recovery. SSE constructions use inverted indices and game-based security definitions; cipher maps use total functions with information-theoretic parameters ($\eta, \varepsilon, \delta$). The approaches address different threat models: SSE hides query results from the server (up to leakage profiles), while cipher maps hide function identity and domain membership through totality and noise closure. Honey encryption [15], which produces plausible-looking plaintexts under wrong keys, shares the intuition that decoding ambiguity provides deniability; cipher maps achieve a similar effect through the noise-decode probability ε .

Garbled circuits and fully homomorphic encryption. Yao’s garbled circuits [22] evaluate arbitrary functions on encrypted inputs via encrypted lookup tables. Gentry’s fully homomorphic encryption [12] enables exact computation on ciphertexts without interaction. Both achieve exact computation on encrypted data. Cipher maps trade exactness for simplicity: a cipher map is a static lookup table (one hash evaluation per query) with tunable error η , whereas garbled circuits require per-gate encryption and are single-use, and FHE incurs substantial computational overhead from noise management in lattice operations.

Homophonic substitution. The representation uniformity property (multiple encodings per value, $K(x) \propto 1/D(x)$) is a modern instance of homophonic substitution [18], in which frequent plaintext symbols are assigned multiple ciphertext equivalents to flatten frequency distributions. The connection is direct: representation uniformity generalizes homophonic substitution from substitution ciphers to arbitrary total functions.

3 The Cipher Map Abstraction

3.1 Preliminaries

Throughout, $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ denotes a cryptographic hash function modeled as a *random oracle* [3]: a truly random function accessible to all parties via oracle queries. Results stated “under the random oracle model” assume h has this property. Elements of any finite type are encoded as bit strings before hashing.

3.2 Definition

Definition 3.1 (Cipher map). Let $f : X \rightarrow Y$ be a *latent function*. A *cipher map* for f is a tuple $(\hat{f}, \text{enc}, \text{dec}, s)$ where:

1. $\hat{f} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a **total function** on n -bit strings;
2. $\text{enc} : X \times \{0, \dots, K(x)-1\} \rightarrow \{0, 1\}^n$ is an encoding function, where $K(x) \geq 1$ is the number of encodings for element x ;
3. $\text{dec} : \{0, 1\}^n \rightarrow Y \cup \{\perp\}$ is a decoding function;
4. s is a secret (the trapdoor) from which \hat{f} , enc , and dec are derived.

The untrusted machine holds \hat{f} . The trusted machine holds enc , dec , and s .

A cipher value $\text{enc}(v, k)$ can be viewed as a cipher map for the constant function $f_v(x) = v$: under this view, cipher values and cipher maps are the same kind of object—a total function on bit strings—simplifying the untrusted machine’s interface.

3.3 Two Construction Strategies

Cipher maps arise from two fundamentally different construction strategies:

Batch construction. The trusted machine invests computation upfront to find a seed s such that \hat{f} satisfies correctness constraints for all $x \in X$ simultaneously. This gives tunable correctness parameter η and optimal space, but requires knowing X and f at construction time. The entropy cipher map (§6) is the general batch construction.

Online construction. The cipher map \hat{f} is defined directly by the hash structure—no seed search is needed. The secret s is a hash key, not a searched seed. Operations are limited to those the hash structure supports algebraically (e.g., set union, intersection). Space and accuracy trade-offs are fixed by the hash width n . The trapdoor Boolean algebra (§9.3) is the online construction.

Both strategies produce objects satisfying Definition 3.1.

3.4 Construction Layers

The four properties (defined in §4) arise from three orthogonal type transformations applied to the latent function.

Layer 1: Undefined injection. Extend the domain X to $X \cup \{\perp_1, \dots, \perp_N\}$ by adding N undefined elements. A function $f : X \rightarrow Y$ lifts to a partial function \tilde{f} that is undefined on the \perp_i . Real elements become a fraction $|X|/(|X| + N)$ of the extended type. This controls the space parameter ε (the probability that random bits form a valid codeword).

Layer 2: Noise closure. Lift the partial function \tilde{f} to a total function \hat{f} by mapping undefined inputs to random outputs via cryptographic hash. The untrusted machine cannot distinguish real inputs from undefined inputs—both produce n -bit output. This yields **Property 1 (Totality)**.

Layer 3: Multiple representations. Replace each value x with $K(x) \geq 1$ distinct representations, keyed by the secret s . Representational equality implies value equality, but value equality does not imply representational equality. Assigning $K(x) \propto 1/D(x)$ equalizes frequencies. This yields **Property 2 (Representation Uniformity)**.

The combined type is cipher(noise(undef(X)))—an element with multiple representations, total evaluation, and diluted domain. Properties 3 (Correctness) and 4 (Composability) are emergent: correctness depends on the construction method; composability depends on the totality guarantee enabling chained evaluation.

These layers are conceptual scaffolding explaining *why* each property exists, not algebraic monads. Whether the layers satisfy formal monad laws in the approximate setting is an open question (see §9).

4 Four Properties

A cipher map $(\hat{f}, \text{enc}, \text{dec}, s)$ for $f : X \rightarrow Y$ is characterized by four properties parameterized by $(\eta, \varepsilon, \mu, \delta)$ (summarized in Table 1).

4.1 Property 1: Totality

Definition 4.1 (Totality). The function $\hat{f} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is *total*: for every $c \in \{0, 1\}^n$, $\hat{f}(c) \in \{0, 1\}^n$. For c chosen uniformly at random from $\{0, 1\}^n \setminus \text{Im}(\text{enc})$, the distribution of $\hat{f}(c)$ is uniform over $\{0, 1\}^n$ under the random oracle model.

Totality ensures that the untrusted machine cannot distinguish a real query $\text{enc}(x, k)$ from a random filler string $c \leftarrow \{0, 1\}^n$, because both produce n -bit outputs. If out-of-domain queries returned an error or \perp , the untrusted machine could identify real queries by observing which inputs produce valid output.

4.2 Property 2: Representation Uniformity (δ -bounded)

Definition 4.2 (Representation uniformity). Let D be a distribution on X . Define the induced distribution on cipher values:

$$Q(c) = \sum_{x \in X} D(x) \cdot \frac{|\{k : \text{enc}(x, k) = c\}|}{K(x)}. \quad (1)$$

The cipher map has δ -representation uniformity if

$$d_{\text{TV}}(Q, \text{Uniform}(\{0, 1\}^n)) \leq \delta, \quad (2)$$

where d_{TV} is total variation distance.

Total variation distance has the operational interpretation that no statistical test can distinguish Q from uniform with advantage exceeding δ . To achieve small δ , assign $K(x) \propto 1/D(x)$ encodings per value (homophonic substitution), so that frequent values get more representations.

Remark 4.1 (Marginal uniformity only). Representation uniformity bounds the *marginal* distribution of individual cipher values. It says nothing about *joint* distributions: an adversary observing sequences of cipher values can still detect correlations. See §8.1 for the encoding granularity trade-off.

4.3 Property 3: Correctness (η -bounded)

Definition 4.3 (Correctness). For x chosen uniformly from X and k chosen uniformly from $\{0, \dots, K(x) - 1\}$:

$$\Pr_{x,k}[\text{dec}(\hat{f}(\text{enc}(x, k))) \neq f(x)] \leq \eta. \quad (3)$$

The parameter $\eta \in [0, 1]$ is a *construction* parameter: it reflects how many constraints the seed s must satisfy. Tolerating $\eta > 0$ makes the seed search faster and the structure smaller. Once built, the failing elements are deterministic for a given seed—the same elements always fail.

Remark 4.2 (Approximation as privacy). Nonzero η also provides privacy: a false positive creates plausible deniability, since the untrusted machine cannot distinguish “element is in the set” from “cipher map erred.” With $\eta = 0$, the answer is deterministic; with $\eta > 0$, each answer carries ambiguity.

4.4 Property 4: Composability

Definition 4.4 (Composability). The composition of two cipher maps is again a cipher map. Specifically: let $(\hat{f}, \text{enc}_f, \text{dec}_f, s_f)$ be a cipher map for $f : X \rightarrow Y$ and $(\hat{g}, \text{enc}_g, \text{dec}_g, s_g)$ a cipher map for $g : Y \rightarrow Z$. Then $\hat{g} \circ \hat{f} : \{0, 1\}^n \rightarrow \{0, 1\}^n$, defined by $(\hat{g} \circ \hat{f})(c) = \hat{g}(\hat{f}(c))$, with encoding enc_f and decoding dec_g , is a cipher map for $g \circ f : X \rightarrow Z$.

Composability is enabled by totality: since \hat{f} is total, its output is always a valid n -bit input to \hat{g} . The untrusted machine can chain cipher maps without needing to decode intermediate results.

Theorem 4.1 (Composition correctness). *Under the hypotheses of Definition 4.4, if \hat{f} has correctness η_f and \hat{g} has correctness η_g , then $\hat{g} \circ \hat{f}$ has correctness*

$$\eta_{g \circ f} \leq \eta_f + \eta_g - \eta_f \eta_g = 1 - (1 - \eta_f)(1 - \eta_g),$$

with equality under independent errors (see §7, Theorem 7.2).

Remark 4.3 (Noise under composition). Noise input to \hat{f} produces random output (by totality). With probability ε , that random output happens to be a valid codeword for \hat{g} . We do not try to control noise output—the cipher map is just a hash. This is a feature: valid-looking outputs do not prove valid inputs.

4.5 Parameter Decomposition

Table 1: Cipher map parameters.

Parameter	Range	Controls	Determined by
η (correctness)	$[0, 1)$	Fraction of wrong answers	Construction (seed search)
ε (noise decode)	$(0, 1)$	$\Pr[\text{random bits valid}]$	Encoding scheme
μ (value cost)	$(0, \infty)$	Bits per element for values	$\mu = H(Y)$
δ (uniformity)	$[0, 1]$	TV distance from uniform	Multiplicity $K(x)$
$K(x)$ (multiplicity)	$\{1, 2, \dots\}$	Encodings per element	Set to $\propto 1/D(x)$
p (entanglement)	$\{1, \dots, k\}$	Encoding granularity	Correlation structure (§8.1)

The space per element decomposes as:

$$\text{bits/element} = -\log_2 \varepsilon + \mu = -\log_2 \varepsilon + H(Y),$$

where $-\log_2 \varepsilon$ bits distinguish signal from noise and $H(Y)$ bits encode the function value. This decomposition is information-theoretic.

5 The Trusted/Untrusted Machine Model

The four properties take operational meaning through a two-party model.

Definition 5.1 (Trusted machine T). The trusted machine T holds: the seed s (trapdoor), the encoding function enc , the decoding function dec , and knowledge of which queries are real vs. filler.

T can:

1. Encode plaintext values: $x \mapsto \text{enc}(x, k)$ for chosen k .
2. Decode cipher results: $r \mapsto \text{dec}(r)$.
3. Inject noise: generate random $c \leftarrow \{0, 1\}^n$ as filler queries.
4. Verify results: check $\text{dec}(\hat{f}(\text{enc}(x, k))) = f(x)$.
5. Reseed: construct a new cipher map with fresh s' when leakage accumulates.

Definition 5.2 (Untrusted machine U). The untrusted machine U holds: the cipher map \hat{f} (a total function on bit strings) and a collection of cipher values (encodings + filler, indistinguishable to U).

U can:

1. Evaluate cipher maps: $c \mapsto \hat{f}(c)$ for any $c \in \{0, 1\}^n$.
2. Return results to T .

U cannot:

1. Decode cipher values (does not hold s or dec).
2. Distinguish real encodings from filler (by Property 1).
3. Determine the domain X or function f (by Property 2, cipher values are δ -close to uniform).
4. Enumerate which inputs are “in-domain” (by totality, every input produces output).

Protocol. The information flow is:

1. T encodes inputs: $\text{enc}(x_1, k_1), \dots, \text{enc}(x_m, k_m)$.
2. T generates filler: $c_1, \dots, c_r \leftarrow \{0, 1\}^n$.
3. T sends all cipher values (real + filler, shuffled) to U .
4. U evaluates \hat{f} on each cipher value and returns results.
5. T decodes the results it cares about; discards filler results.

What U observes is a sequence of n -bit strings as input and a sequence of n -bit strings as output. Table 2 summarizes the guarantees. Under the four properties:

- **Totality:** every input produces output; no error signals leak domain information.
- **Representation uniformity:** cipher values are δ -close to uniform; frequency analysis is bounded by δ .
- **Correctness:** irrelevant to U (only T decodes).
- **Composability:** U can chain cipher maps without decoding intermediate results.

Table 2: What the four properties guarantee against U .

Adversary capability	Prevented by	Mechanism
Distinguish real from filler	Totality	All queries produce output
Frequency analysis	Rep. uniformity	Values δ -close to uniform
Learn function values	Trapdoor	Cannot invert h without s
Enumerate domain X	Totality + trapdoor	Cannot test membership
Detect query correlations	Not fully	Only marginal uniformity

The last row is the honest limitation: the four properties do not provide full correlation hiding unless the encoding granularity encompasses the correlated values (§8.1).

6 The Batch Construction

The batch construction requires knowing X and f at construction time and invests computation in a seed search to find a cipher map satisfying correctness constraints for all stored elements. This section develops the batch construction from the information-theoretic lower bound through the acceptance predicate framework, culminating in the entropy cipher map—the general batch cipher map for arbitrary $f : X \rightarrow Y$.

6.1 Information-Theoretic Lower Bound

Theorem 6.1 (Lower bound). *Any data structure representing an approximate map \hat{f} for $f : X \rightarrow Y$ with noise decode probability ε and correctness $\eta = 0$ over n stored elements requires at least*

$$n(-\log_2 \varepsilon + H(Y)) \text{ bits}, \tag{4}$$

or equivalently $-\log_2 \varepsilon + H(Y)$ bits per element.

Proof. The bound decomposes into two independent information requirements.

Membership component. Consider the membership predicate $\text{dec}(\hat{f}(c)) \neq \perp$. For a universe U with $|U| \gg n$, the structure must distinguish n stored elements from $|U| - n$ non-elements, with at most an ε fraction of non-elements decoding to valid output. There are $\binom{|U|}{n}$ possible sets of size n from a universe U . Any representation must distinguish all such sets, requiring at least $\log_2 \binom{|U|}{n}$ bits. For $|U| \gg n$, Stirling's approximation gives $\log_2 \binom{|U|}{n} \approx n \log_2(|U|/n)$ bits, or $\log_2(|U|/n)$ bits per element. Setting $\varepsilon = n/|U|$ (the fraction of the universe that decodes validly), this is $n \log_2(1/\varepsilon)$ bits.

Value component. Independently of membership, the structure must encode the function values $(f(x_1), \dots, f(x_n))$. By Shannon's source coding theorem [17], encoding n independent draws from the distribution (p_y) requires at least $nH(Y)$ bits.

Since membership and value encoding are jointly necessary, the total requirement is $n(-\log_2 \varepsilon + H(Y))$ bits. The additivity holds because the membership bits and value-encoding bits convey independent information: by the entropy chain rule, $H(\text{membership}, \text{value}) = H(\text{membership}) + H(\text{value} \mid \text{membership})$, and conditioning on membership (i.e., knowing which elements are stored) does not reduce the entropy of the value assignment, since the values $(f(x_1), \dots, f(x_n))$ are an independent degree of freedom. \square

6.2 Acceptance Predicates

The construction depends on a rule for deciding whether a hash value encodes a particular output. We abstract this as an *acceptance predicate*.

Definition 6.1 (Acceptance predicate). An acceptance predicate for codomain Y is a family of disjoint sets $\{A(y) \subseteq \{0, 1\}^n : y \in Y\}$. The decoder is $\text{dec}(\text{hash}) = y$ if $\text{hash} \in A(y)$, and \perp otherwise. The acceptance probability for value y is $\alpha(y) = |A(y)|/2^n$, and the noise-decode probability is $\varepsilon = \sum_y \alpha(y) = |\bigcup_y A(y)|/2^n$.

Remark 6.1 (XOR-scrambled decode). The decode function applies a secret-derived XOR mask before checking acceptance boundaries: $\text{dec}(r)$ checks whether $r \oplus s_{\text{key}}$ falls in the acceptance region for some y . Without the scramble key, the mapping from hash values to output values is pseudorandom; contiguous acceptance regions in the scrambled space appear scattered in the raw hash space. This prevents an adversary observing hash outputs from inferring which outputs correspond to which values, even if the acceptance regions are contiguous.

Different predicates give different space and construction-time trade-offs:

- **Prefix-free code.** $A(y) = \{c \in \{0, 1\}^n : c \text{ starts with } \text{codeword}(y)\}$. Acceptance probability $\alpha(y) = 2^{-|\text{code}(y)|}$. Shannon-optimal when $|\text{code}(y)| \approx -\log_2 p_y$. This is the entropy cipher map construction.
- **Threshold.** $A(y) = \{c : c < t(y)\}$ for thresholds $0 = t_0 < t_1 < \dots$. Acceptance probability $\alpha(y) = (t_{y+1} - t_y)/2^n$. Allows fine-grained control of per-value acceptance rates.

- **Range partition.** Partition $[0, 2^n)$ into contiguous ranges $[l_y, l_y + w_y)$. Setting $w_y \propto p_y$ achieves Shannon-optimal allocation with uniform acceptance probability.

All three achieve the same space bound $-\log_2 \varepsilon + H(Y)$ bits per element when acceptance probabilities are Shannon-optimal ($\alpha(y) \propto p_y$). They differ in construction time and implementation simplicity.

Shannon-optimal allocation and frequency hiding. Setting $\alpha(y) \propto \Pr[f(X) = y]$ simultaneously achieves two goals. First, it minimizes space: the per-element encoding cost $-\log_2 \alpha(y) \approx -\log_2 p_y$ is the Shannon information content of value y , and the average cost $\sum_y p_y (-\log_2 p_y) = H(Y)$ is the entropy of the output distribution. Second, it hides value frequencies: noise queries (random hash values) hit acceptance set $A(y)$ with probability $\alpha(y) \propto p_y$ —the same distribution as real queries. An adversary observing output values cannot distinguish whether a result came from a real query or noise, because the output frequency profile is identical in both cases.

These are not two independent optimizations that happen to coincide. Shannon-optimal coding IS frequency hiding: matching the acceptance distribution to the output distribution simultaneously minimizes space and maximizes indistinguishability. Any deviation from $\alpha(y) \propto p_y$ both wastes space (encoding rare values with fewer bits than their information content requires) and leaks information (the frequency profile of outputs differs between real and noise queries).

6.3 Construction Algorithm

Algorithm 1 Batch Cipher Map Construction

```

1: function BUILDCIPHERMAP( $M, A, h, \eta$ )      ▷  $A$ : acceptance predicate,  $\eta$ : target error rate
2:    $m \leftarrow |M|$ 
3:    $\ell \leftarrow 0$ 
4:   while true do
5:     Shuffle( $M$ )                               ▷ random order for unbiased early stopping
6:     fail  $\leftarrow$  0; pass  $\leftarrow$  0
7:     for  $(x, y) \in M$  do
8:       hash  $\leftarrow h(\ell) \oplus h(x)$ 
9:       if hash  $\in A(y)$  then
10:        pass  $\leftarrow$  pass + 1
11:        if pass  $\geq \lceil (1 - \eta)m \rceil$  then
12:          return CipherMap( $h, A, \ell$ )          ▷ early accept
13:        end if
14:      else
15:        fail  $\leftarrow$  fail + 1
16:        if fail  $> \lfloor \eta m \rfloor$  then
17:          break                                  ▷ early reject
18:        end if
19:      end if
20:    end for
21:     $\ell \leftarrow \ell + 1$ 
22:  end while
23: end function

```

Algorithm 1 searches for a seed ℓ such that at most $\lfloor \eta \cdot m \rfloor$ elements of M fail the acceptance predicate. When $\eta = 0$, the first failure triggers rejection (the exact case). Query evaluation computes $\hat{f}(c) = h(\ell) \oplus h(c)$ and decodes via A .

Two early-stopping rules avoid checking all m elements:

- **Early reject:** if failures exceed $\lfloor \eta m \rfloor$, no remaining successes can help.
- **Early accept:** if successes reach $\lceil (1 - \eta)m \rceil$, the error budget is satisfied regardless of remaining elements.

Shuffling M each iteration ensures both rules trigger as early as possible: failures are not clustered at the end by a fixed ordering.

6.4 Space Optimality

Theorem 6.2 (Space complexity). *The batch construction achieves space complexity*

$$(1 - \eta)(-\log_2 \varepsilon + \mu) \text{ bits per element}, \quad (5)$$

where $\mu = H(Y)$ is the mean encoding length. When $\eta = 0$, this matches the lower bound of Theorem 6.1.

Proof. Step 1: False negative reduction. When false negatives are permitted at rate η , only $(1 - \eta)n$ of the n elements must decode correctly, reducing the effective element count.

Step 2: Per-element bit cost. For each stored element x_i , the hash $h(\ell) \oplus h(x_i)$ must fall in the acceptance set $A(y_i)$ (Definition 6.1). Under the random oracle model, this occurs with probability $\alpha(y_i)$. Shannon-optimal allocation sets $\alpha(y_i) \propto p_{y_i}$, giving per-element information cost $-\log_2 \alpha(y_i) \approx -\log_2 p_{y_i}$ and expected cost per element $\sum_y p_y (-\log_2 \alpha(y)) = \mu = H(Y)$.

Step 3: False positive control. For non-stored elements $x \notin M$, the hash $h(\ell) \oplus h(x)$ is uniformly distributed under the random oracle model. The probability of falling in *any* acceptance set is $\varepsilon = \sum_y \alpha(y)$, the noise-decode probability. Achieving target rate ε requires $-\log_2 \varepsilon$ bits of hash beyond the value encoding.

Step 4: Combining. Each stored element costs $-\log_2 \varepsilon + \mu$ bits, and $(1 - \eta)n$ elements are stored, giving total space $(1 - \eta)n(-\log_2 \varepsilon + \mu)$ bits. \square

Corollary 6.3 (Asymptotic optimality). *When $\eta = 0$, the batch construction achieves $-\log_2 \varepsilon + H(Y)$ bits per element, matching the information-theoretic lower bound.*

6.5 Construction Time and Bucketing

Proposition 6.4 (Per-seed success probability). *For m stored elements with acceptance predicate A and target error rate η , each element independently passes with probability $\alpha(y_i)$ under the random oracle model. The number of failures F follows a Poisson binomial distribution, and the seed is accepted when $F \leq \lfloor \eta m \rfloor$.*

Special cases:

- $\eta = 0$: $p = \prod_{i=1}^m \alpha(y_i)$ (all must pass).
- Uniform α : $p = \Pr[\text{Bin}(m, 1 - \alpha) \leq \lfloor \eta m \rfloor]$.

The expected number of seeds tried is $1/p$.

Proof. Under the random oracle model, the hash values $h(\ell) \oplus h(x_i)$ for distinct x_i are independent. Element x_i fails with probability $1 - \alpha(y_i)$. The failure count $F = \sum_i \mathbf{1}[\text{hash}_i \notin A(y_i)]$ is a sum of independent Bernoulli variables. The seed is accepted when $F \leq \lfloor \eta m \rfloor$, giving the Poisson binomial CDF. For uniform α , this reduces to the binomial CDF. \square

The effect of $\eta > 0$ on construction time is dramatic. For uniform α and $m = 100$:

α	$\eta = 0$	$\eta = 0.01$	$\eta = 0.05$
0.5	2^{100}	2^{93}	2^{74}
0.9	3.8×10^4	3.1×10^3	17
0.99	2.7	1.5	1.0

When η exceeds the expected failure rate $1 - \alpha$, the binomial tail concentrates and construction becomes near-certain on the first seed. The transition is sharp: a few percent of tolerable error can reduce construction time by tens of orders of magnitude.

Bucketed construction. The intractability of the global seed search is not inherent. Partition the m elements into k buckets (e.g., by $h(x) \bmod k$) and find a separate seed ℓ_j for each bucket.

Proposition 6.5 (Bucketed construction time ($\eta = 0$)). *With k buckets of expected size m/k and $\eta = 0$, the expected total construction time is*

$$T(k) = k \cdot \left(\frac{1}{\bar{\alpha}}\right)^{m/k}, \quad (6)$$

where $\bar{\alpha} = (\prod_i \alpha(y_i))^{1/m}$ is the geometric mean acceptance probability.

Proof. Each bucket has m/k elements (in expectation). The per-bucket success probability is $\bar{\alpha}^{m/k}$ by Proposition 6.4. Expected trials per bucket: $(1/\bar{\alpha})^{m/k}$. Summing over k independent buckets gives the result. \square

The trade-off between construction time and space is controlled by k :

Buckets k	Seeds stored	Construction time	Space overhead
1 (global)	1	$(1/\bar{\alpha})^m$	$O(\log m)/m$ bits/elem
\sqrt{m}	\sqrt{m}	$\sqrt{m} \cdot (1/\bar{\alpha})^{\sqrt{m}}$	$O(\log m/\sqrt{m})$ bits/elem
m (per-element)	m	$m/\bar{\alpha}$ (linear)	$O(\log(1/\bar{\alpha}))$ bits/elem

At $k = m$, each bucket has one element, construction is linear in m , and the overhead is $\log_2(1/\bar{\alpha})$ bits per element for the per-element seed—the same order as the hash width. This is the regime of practical perfect hash functions [2].

An alternative construction uses perfect hash functions (PHFs) to assign elements to slots in $O(m)$ time, avoiding the seed search entirely. A PHF maps the m cipher keys to m distinct slots; each slot stores the XOR-masked value encoding. Construction time drops from exponential in bucket size to linear in m . A reference implementation using a RecSplit-family PHF achieves 700 documents per second on the 20 Newsgroups corpus (18,266 documents, 58,903 words), compared to approximately 10 documents per second with the seed search construction.

6.6 Instantiations

6.6.1 Entropy Cipher Map

The *entropy cipher map* is the batch construction (Algorithm 1) instantiated with a prefix-free acceptance predicate. It is the general batch cipher map for arbitrary $f : X \rightarrow Y$, achieving Shannon-entropy-optimal value encoding.

Latent function. Arbitrary $f : X \rightarrow Y$ where Y is finite.

Construction. Assign a prefix-free code $C_y \subseteq \{0, 1\}^*$ for each $y \in Y$, where the fraction of hash space covered by C_y is proportional to $\Pr_D[f(X) = y]$. Find a seed s (via two-level hashing, §6.5) such that

$$h(x||s) \in C_{f(x)} \quad \text{for all } x \in X.$$

Evaluation: compute $h(c||s)$ and decode via the prefix-free code.

Two-level hash (practical algorithm).

1. A primary hash maps x to a bucket index $j \in \{0, \dots, b-1\}$.
2. For each bucket, find a local seed s_j such that $h(x||s_j)$ decodes to $f(x)$ for all x in bucket j .
3. Store s_j in a seed table $H[j]$.
4. Query: compute bucket j (1 hash), evaluate $h(x||H[j])$ and decode (1 hash). Total: $O(1)$ hash operations.

Mapping to cipher map abstraction.

- $\hat{f}(c) = h(c||H[\text{bucket}(c)])$, total function.
- $\text{enc}(x, 0) = x$ (simplest version; $K(x) = 1$).
- $\text{dec}(r) = y$ if $r \in C_y$ for some y , else \perp .

Parameter instantiation.

Parameter	Value	Justification
η	Tunable	Fraction of elements failing acceptance predicate
ε	Code-dependent	$\Pr[\text{random bits} \in \bigcup_y C_y]$
μ	$H(Y)$	Shannon entropy of output distribution
δ	From code assignment	If $ C_y \propto \Pr[f(X) = y]$, approaches uniform
Space	$-\log_2 \varepsilon + H(Y)$ bits/element	

Property status. Totality: **yes**. Representation uniformity: **achievable** (by assigning multiple codes proportional to output frequency). Correctness: **yes** ($\eta \approx 0$, tunable via Algorithm 1). Composability: **yes** (output is an n -bit string; serves as input to another cipher map).

Proposition 6.6 (Entropy cipher map space). *The space per element of an entropy cipher map for $f : X \rightarrow Y$ with output distribution $(p_y)_{y \in Y}$ equals $-\log_2 \varepsilon + H(Y)$ bits, where $H(Y) = -\sum_y p_y \log_2 p_y$.*

Proof. The term $-\log_2 \varepsilon$ is the per-element cost of distinguishing signal from noise: each stored element requires that its hash prefix match a valid codeword, which occurs with probability ε for random inputs. The term $H(Y)$ is the expected code length under Shannon-optimal prefix-free coding, which equals the entropy of the output distribution by Shannon’s source coding theorem [17]. These two components are additive because the noise-rejection bits and the value-encoding bits occupy disjoint portions of the hash output. \square

Remark 6.2 (Set membership as a cipher map). Set membership $\mathbf{1}_A : X \rightarrow \text{Bool}$ is a cipher map with $Y = \{\text{true}, \text{false}\}$. The acceptance predicate partitions hash space into $A(\text{true})$ and $A(\text{false})$, with both outputs appearing as opaque bit strings. Shannon-optimal allocation sets $|A(\text{true})| \propto \Pr[x \in A]$ and $|A(\text{false})| \propto \Pr[x \notin A]$. For members, the seed search ensures $h(\ell) \oplus h(x) \in A(\text{true})$. For non-members, the hash lands in $A(\text{false})$ with probability $1 - \varepsilon$, providing both correct classification and output-frequency hiding. Space per element: $-\log_2 \varepsilon + H(\text{Bool})$ bits, where $H(\text{Bool}) = -p \log_2 p - (1-p) \log_2 (1-p)$ for $p = \Pr[x \in A]$.

The traditional Bloom filter [6] tests membership but reveals the raw Boolean result. The cipher map version hides both the input (via enc) and the output (via the acceptance predicate), at the cost of storing per-element seeds and accepting a noise-decode probability ε .

7 Composition

7.1 Warm-Up: The AND Gate

Before proving the general composition theorem, we analyze a concrete case: the AND gate operating on two independent Bernoulli Boolean values B_1, B_2 with correctness probabilities $p_1 = 1 - \eta_1$ and $p_2 = 1 - \eta_2$.

Proposition 7.1 (AND gate correctness). *The correctness probability of $\text{AND}(B_1, B_2)$ as an approximation of $\text{AND}(x_1, x_2)$ depends on the latent inputs:*

x_1	x_2	$\text{AND}(x_1, x_2)$	$\Pr[\text{correct}]$
1	1	1	$p_1 p_2$
1	0	0	$1 - p_1(1 - p_2)$
0	1	0	$1 - p_2(1 - p_1)$
0	0	0	$p_1 + p_2 - p_1 p_2$

Proof. We verify the (1, 0) case; the others are analogous. When $x_1 = 1, x_2 = 0$, the correct output is 0. The output is wrong (= 1) only when $B_1 = 1$ (correct, probability p_1) and $B_2 = 1$ (incorrect, probability $1 - p_2$). So $\Pr[\text{wrong}] = p_1(1 - p_2)$ and $\Pr[\text{correct}] = 1 - p_1(1 - p_2) = 1 - p_1 + p_1 p_2$. \square

The output is a *4th-order Bernoulli Boolean*: four distinct correctness probabilities despite the Boolean type having only two values. Rather than tracking all four, we can bound them by an interval $[\min_{\text{cases}} p_{\text{correct}}, \max_{\text{cases}} p_{\text{correct}}]$. For AND with $p_1, p_2 \in (0.5, 1)$: minimum $p_1 p_2$ (case (1, 1)); maximum $p_1 + p_2 - p_1 p_2$ (case (0, 0)).

7.2 General Composition Theorem

Theorem 7.2 (Composition correctness). *Let \hat{f} be a cipher map for $f : X \rightarrow Y$ with correctness η_f , and \hat{g} a cipher map for $g : Y \rightarrow Z$ with correctness η_g . Then $\hat{g} \circ \hat{f}$ is a cipher map for $g \circ f$ with correctness*

$$\eta_{g \circ f} \leq \eta_f + \eta_g - \eta_f \eta_g = 1 - (1 - \eta_f)(1 - \eta_g). \quad (7)$$

Equality holds when \hat{f} and \hat{g} use independent seeds and \hat{g} 's correctness on $\hat{f}(c)$ is independent of whether \hat{f} was correct (the re-randomization condition).

Proof. The composition is incorrect when at least one map errs. Let $A = \{\hat{f} \text{ errs}\}$ and $B = \{\hat{g} \text{ errs}\}$. By inclusion-exclusion, $\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$.

Loose bound. Dropping $\Pr[A \cap B] \geq 0$ gives the union bound $\eta_{g \circ f} \leq \eta_f + \eta_g$.

Under re-randomization. If the errors are independent ($\Pr[A \cap B] = \eta_f \eta_g$), inclusion-exclusion gives $\eta_{g \circ f} = \eta_f + \eta_g - \eta_f \eta_g = 1 - (1 - \eta_f)(1 - \eta_g)$.

In practice, \hat{f} produces a specific encoding of $f(x)$, not a uniformly random one. If \hat{g} 's correctness depends on which encoding it receives, the errors may be positively correlated ($\Pr[A \cap B] \leq \eta_f \eta_g$), making the bound $\eta_f + \eta_g - \eta_f \eta_g$ conservative. For specific circuits, the case-by-case interval arithmetic from §7.1 gives tighter bounds. \square

Remark 7.1 (Garbage propagation). When \hat{f} produces an incorrect output on some input, the value fed to \hat{g} is effectively random. Since \hat{g} is a total function, it produces *some* output on this random input, which is correct with probability at most ε_g (the noise-decode probability of \hat{g}). Thus the bound $\eta_{\text{total}} \leq \eta_f + \eta_g - \eta_f \eta_g$ is an upper bound; the actual error rate may be marginally lower due to accidental correctness on garbage inputs. For small η_f, η_g , this distinction is negligible.

7.3 Composition Chains

Corollary 7.3 (Chain composition). *For a chain of m cipher maps $\hat{f}_1, \dots, \hat{f}_m$:*

$$\eta_{\text{total}} = 1 - \prod_{i=1}^m (1 - \eta_i). \quad (8)$$

If all $\eta_i = \eta$:

$$\eta_{\text{total}} = 1 - (1 - \eta)^m \approx m\eta \quad \text{for small } \eta. \quad (9)$$

Proof. By induction on Theorem 7.2. The base case $m = 2$ is the theorem. For the inductive step, the composition of $m - 1$ maps has correctness $1 - \prod_{i=1}^{m-1} (1 - \eta_i)$; composing with \hat{f}_m gives $1 - \prod_{i=1}^{m-1} (1 - \eta_i) \cdot (1 - \eta_m) = 1 - \prod_{i=1}^m (1 - \eta_i)$. The approximation follows from $\ln(1 - \eta) \approx -\eta$ for small η , giving $(1 - \eta)^m \approx e^{-m\eta} \approx 1 - m\eta$. \square

Example 7.1. A circuit of 100 cipher maps each with $\eta = 10^{-6}$: $\eta_{\text{total}} = 1 - (1 - 10^{-6})^{100} \approx 10^{-4}$.

7.4 Error Accumulation by Gate Type

For specific circuits, the AND gate analysis (Proposition 7.1) can be extended to OR, NOT, and arbitrary Boolean gates. Each gate type induces a different case table. Rather than tracking exact correctness per case, interval arithmetic propagates bounds: at each gate, compute $[\eta_{\min}, \eta_{\max}]$ from the input intervals. This gives input-dependent bounds tighter than the worst-case Theorem 7.2.

8 Representation Uniformity and Encoding Granularity

We measure representation uniformity using total variation distance d_{TV} , which bounds the distinguishing advantage of any statistical test. We use TV rather than max-divergence (D_∞) because the constructions are designed around frequency equalization, which naturally yields TV bounds.

8.1 The Encoding Granularity Principle

Representation uniformity is defined *per cipher map*: the *encoding granularity* is the domain type X , and uniformity applies to the distribution over encodings of individual elements of X . What correlations are hidden depends on what is encoded as a unit. Consider encrypting a pair of Boolean values (a, b) where a and b are correlated.

Component-wise encoding ($p = 1$). Encode a and b as separate cipher values. Each component's marginal distribution is δ -close to uniform, but joint correlations leak.

Pair-wise encoding ($p = 2$). Encode (a, b) as a single cipher value. The distribution over pair encodings is δ -close to uniform; correlations between a and b are hidden. Cost: domain is $\{0, 1\}^2$ (4 elements) rather than $\{0, 1\}$ (2 elements); space per encoding grows.

Whole-state encoding ($p = k$). Encode the entire program state as one cipher value. All correlations are hidden. Cost: domain is the full state space, typically prohibitive.

Definition 8.1 (Entanglement parameter). For a system encoding k correlated Boolean values, the *entanglement parameter* p is the number of values encoded as a single unit. The system has type cipher($\{0, 1\}^p$) $^{k/p}$, where each block of p bits is encoded jointly.

Proposition 8.1 (Granularity and privacy). Let \hat{f} be a cipher map for $f : X_1 \times X_2 \rightarrow Y$ with representation uniformity δ . Then the marginal distribution of $\text{enc}((x_1, x_2), k)$ over random $(x_1, x_2) \sim D$ and random k is δ -close to uniform, and the joint distribution of (x_1, x_2) is hidden up to δ .

Conversely, let \hat{f}_1, \hat{f}_2 be independent cipher maps for $f_1 : X_1 \rightarrow Y_1$ and $f_2 : X_2 \rightarrow Y_2$, each with representation uniformity δ . Then each marginal cipher value is δ -close to uniform, but correlations between x_1 and x_2 are **not hidden**, even when $\delta = 0$.

Proof. The first part follows directly from Definition 4.2 applied to the joint cipher map. For the second part: even with $\delta = 0$ (perfect marginal uniformity), enc_i is injective for each fixed k_i . The joint distribution of $(\text{enc}_1(x_1, k_1), \text{enc}_2(x_2, k_2))$ is therefore a bijective image of the joint distribution of $((x_1, k_1), (x_2, k_2))$, preserving all statistical dependence between x_1 and x_2 . An adversary observing N draws can estimate this joint distribution and recover the correlation structure of (x_1, x_2) . \square

8.2 Compositional Leakage

Even when each cipher map has uniform marginal output, composing multiple cipher maps on a shared input leaks the correlation structure of the latent functions. Consider cipher maps $\hat{f}_1 : \{0, 1\}^n \rightarrow \{0, 1\}^n$ and $\hat{f}_2 : \{0, 1\}^n \rightarrow \{0, 1\}^n$ for latent $f_1 : X \rightarrow Y$ and $f_2 : X \rightarrow Z$, each with Shannon-optimal acceptance predicates. Individually, the outputs \hat{y} and \hat{z} are uniform. But the untrusted machine observes both $\hat{y} = \hat{f}_1(c)$ and $\hat{z} = \hat{f}_2(c)$ for the same cipher value c . Since \hat{f}_1 and \hat{f}_2 are deterministic, the pair (\hat{y}, \hat{z}) is a deterministic function of c , and the joint distribution of (\hat{y}, \hat{z}) preserves the latent correlation between f_1 and f_2 (Proposition 8.1).

If a third cipher map \hat{f}_3 takes this correlated pair as input, its output inherits the non-uniformity: \hat{f}_3 was constructed assuming a design distribution over its inputs, but the actual input distribution reflects latent correlations. The output \hat{w} is non-uniform even if \hat{f}_3 's acceptance predicate is Shannon-optimal for the marginal distribution of W . Every observable correlation between cipher values carries information about the latent data.

Mitigation strategies. Two approaches exist, both costly:

Build the composition directly. Construct a single cipher map for the composed function $g(x) = f_3(f_1(x), f_2(x))$ as one batch construction. The untrusted machine sees only the cipher input \hat{x} and the final cipher output \hat{w} ; no intermediate values are exposed. This eliminates compositional leakage entirely but costs $O(|X| \cdot (-\log_2 \varepsilon + H(W)))$ space and exponential construction time—the entire program is a single lookup table.

Compose components and inject noise. Keep the practical component-wise evaluation ($\hat{f}_1, \hat{f}_2, \hat{f}_3$ as separate cipher maps) and add R noise queries for every N real queries. By totality, noise and real queries are indistinguishable to U . The adversary's estimates of pairwise correlations have variance $O(1/N)$; mixing with $R \gg N$ noise pairs dilutes the signal proportionally. This costs bandwidth rather than space.

The fundamental trade-off: constructing the composition as a single cipher map gives full privacy at the cost of space and construction time. Composing component cipher maps is practical but leaks intermediate correlations. Noise injection is an intermediate strategy that trades bandwidth for privacy. The entanglement parameter p controls the granularity of this trade-off.

Honest limitations.

1. Marginal uniformity says nothing about joint distributions.
2. Compositional leakage is inherent in component-wise evaluation; only joint encoding or noise injection can mitigate it.
3. Equality pattern leakage is fundamental: any deterministic encoding leaks the equality pattern of its inputs. Multiple representations ($K(x) > 1$) reduce but do not eliminate this.

9 Discussion

9.1 Relationship to the Bernoulli Model

The correctness parameter η of a cipher map corresponds directly to the error rate in a Bernoulli approximation. A cipher map with correctness η induces a Bernoulli approximation of the latent function f : for each $x \in X$, $\Pr[\text{dec}(\hat{f}(\text{enc}(x, k))) \neq f(x)] = \eta(x)$, where the failing elements are deterministic for a given seed.

The composition formula $\eta_{g \circ f} = 1 - (1 - \eta_f)(1 - \eta_g)$ is the standard error accumulation for independent Bernoulli events, identical to the formula derived in the noisy gates framework for AND gates over Bernoulli Booleans. This is not coincidental: a cipher map *is* a Bernoulli approximation, viewed through the lens of hash-based total functions.

The Bernoulli framework organizes approximation into a layered type hierarchy: $\text{Bool} \rightarrow \text{Set} \rightarrow \text{Map} \rightarrow \text{Relation} \rightarrow \text{Type}$, where each level inherits the error model from the level below. The

formalism presented here covers maps ($X \rightarrow Y$); relations and algebraic types (sum types, product types) require the extensions developed in the Bernoulli relations and algebraic types work [20]. The BHF (Bernoulli Hash Function) construction from that framework is the optimal cipher map for the batch strategy: it achieves the $-\log_2 \varepsilon + H(Y)$ lower bound with $O(1)$ query time.

9.2 Algebraic Structure

The cipher map abstraction relates to algebraic concepts from the theory of monoid homomorphisms. The decoding function dec is (by construction) a left inverse of enc , and for cipher maps that preserve algebraic structure (e.g., the trapdoor Boolean algebra), dec is a homomorphism from the cipher monoid to the latent monoid—but only up to the equivalence $c_1 \sim c_2 \iff \text{dec}(c_1) = \text{dec}(c_2)$. The quotient $\{0, 1\}^n / \sim$ is isomorphic to the latent structure.

We note this connection but do not develop it further. The category-theoretic framing (cipher functors lifting monoids to cipher monoids) is a natural extension but requires careful treatment of the approximate setting: the functor laws hold exactly on the quotient but only approximately on representatives.

9.3 Online Construction

The batch construction developed in §6 requires knowing the domain X and function f at construction time. An alternative *online* strategy encodes sets incrementally via bitwise OR of per-element hashes, with no seed search. Union, intersection, and complement reduce to bitwise OR, AND, and NOT; union is exact but intersection and complement are approximate (cross-element hash collisions and the pigeonhole principle, respectively). This approach—the *trapdoor Boolean algebra*—is developed in a companion paper [20]. The batch and online strategies offer complementary trade-offs: batch achieves information-theoretically optimal space with tunable correctness; online enables incremental construction and algebraic composition at the cost of space exponential in set size.

9.4 Application: Encrypted Search

Encrypted search—information retrieval over confidential data by an untrusted provider—is a direct application of the cipher map abstraction. The domain-specific vocabulary maps to the general framework as follows.

Search agent = trusted machine T . Holds the trapdoor s , encodes queries, decodes results.

Encrypted search provider = untrusted machine U . Holds cipher maps and cipher values; evaluates $\hat{f}(c)$ for any $c \in \{0, 1\}^n$ without decoding.

Information need = latent function $f : X \rightarrow Y$. For keyword search, $f = \mathbf{1}_A$ (set indicator); for ranked retrieval, f is a scoring function.

Hidden query = cipher value $\text{enc}(x, k)$. The provider evaluates the cipher map on this value without learning x .

Secure index = cipher map \hat{f} stored on the provider.

The four properties specialize to encrypted search:

1. **Totality:** The provider can evaluate the index on any bit string; filler queries produce output indistinguishable from real queries.
2. **Representation uniformity:** Hidden queries are δ -close to uniform; frequency analysis of query traffic is bounded by δ .
3. **Correctness:** The search agent recovers the correct answer with probability $\geq 1 - \eta$; nonzero η provides plausible deniability.
4. **Composability:** Boolean combinations of queries (AND, OR, NOT) compose as cipher map chains with predictable error accumulation.

Experimental validation. A reference implementation (`cipher-maps`, Python) validates the encrypted search application on the 20 Newsgroups corpus (18,266 documents, 58,903 unique words). Construction uses perfect hash functions (via `phobic`) with $n = 8$ -bit cipher Booleans ($p_T = 0.05$, $p_F = 0.90$, $p_N = 0.05$). At 5,000 documents: construction takes 5.9 seconds (843 documents per second), single-term queries achieve perfect recall (1.0) with precision 0.39 (matching the theoretical false positive rate $p_T = 0.05$ per document). Multi-term AND queries improve precision (FP drops from 248 to 12 for 3-term AND) while maintaining perfect recall. OR and NOT queries lose recall (0.97 and 0.88 respectively) due to noise propagation through the cipher Boolean operations. The full 18,266-document index builds in 25.6 seconds.

Boolean search queries composed from AND, OR, and NOT can be evaluated entirely on the untrusted machine using cipher Boolean types. A cipher Boolean type partitions the hash space into True, False, and noise regions (e.g., 5%, 90%, 5% respectively). AND, OR, and NOT are themselves cipher maps over the cipher Boolean space. Members of a cipher set are forced into the True region by construction; non-members land randomly in True (5%, the false positive rate), False (90%), or noise (5%). Experimental validation on 5,000 documents shows perfect recall for AND queries with false positives decreasing from 248 (single term) to 12 (3-term AND). Details are developed in the companion paper on algebraic cipher types [21].

The correctness parameter η has a precise deniability interpretation for Boolean-valued cipher maps.

Proposition 9.1 (Bayesian deniability). *Let \hat{f} be a cipher map for $f : X \rightarrow \{0, 1\}$ with symmetric error rate η (i.e., $\Pr[y = 1 \mid f(x) = 0] = \Pr[y = 0 \mid f(x) = 1] = \eta$), and let $\pi = \Pr[f(x) = 1]$ be the prior. If the adversary observes the decoded output $y = 1$, then*

$$\Pr[f(x) = 1 \mid y = 1] = \frac{\pi(1 - \eta)}{\pi(1 - \eta) + (1 - \pi)\eta}. \quad (10)$$

For $\eta = 0$, the posterior equals 1 (no deniability). For $\eta = 1/2$, the posterior equals π (observation is useless).

Proof. By Bayes' rule: $\Pr[y = 1 \mid f(x) = 1] = 1 - \eta$ (correct positive) and $\Pr[y = 1 \mid f(x) = 0] = \eta$ (false positive). Applying Bayes' theorem with prior π gives the stated formula. \square

9.5 Open Questions

1. **Max-divergence vs. TV distance for δ .** TV distance bounds distinguishing advantage; max-divergence (D_∞) would give per-element bounds. Which is the right metric depends on the threat model.

2. **Tight bounds for intersection and NOT.** The intersection false positive rate from cross-element collisions in the trapdoor Boolean algebra needs a closed-form expression as a function of $|A|$, $|B|$, $|A \cap B|$, and n . For NOT, deriving $\eta_{\text{NOT}}(|A|, n)$ would complete the parameter instantiation.
3. **Adaptive encoding granularity.** Can the entanglement parameter p be made adaptive—encoding pairs only for correlated values while encoding independent values separately?
4. **Noise-to-signal probability under composition.** Through a chain of m cipher maps, what is the probability that a filler query produces a decodable output at the end? Naively $1 - (1 - \varepsilon)^m$, but this assumes independence of the ε events across maps.
5. **Layered construction laws.** Do the three construction layers (undef, noise, cipher) satisfy formal algebraic laws in the approximate setting? If so, this would enable equational reasoning about cipher map constructions.
6. **Sum-type confidentiality.** Encoding $\text{OT}(X + Y)$ (the sum type as a single opaque value) hides which branch was taken but breaks composability, since downstream cipher maps cannot dispatch on the branch without decoding. Encoding $\text{OT}(X) + \text{OT}(Y)$ (separate opaque values with a visible tag) preserves composability but leaks the branch tag to the untrusted machine. Whether this trade-off is inherent requires formal proof.

9.6 What This Framework Is Not

To set clear expectations:

- *Not ORAM.* ORAM hides access patterns by reshuffling storage with polylogarithmic overhead. Cipher maps use static total functions.
- *Not differential privacy.* Differential privacy bounds what can be learned about a *dataset* from a *query answer*. Cipher maps bound what an untrusted evaluator learns about a *function* from *opaque evaluations*.
- *Not simulation-based security.* We do not claim that the untrusted machine’s view can be simulated. The guarantees are parameterized (δ, ε) rather than negligible in a security parameter.

Acknowledgments

The constructions in this paper draw on the author’s earlier work on the Bernoulli data type library and the trapdoor Boolean algebra.

References

- [1] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.
- [2] Djamel Belazzougui, Fabiano C Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *European Symposium on Algorithms*, pages 682–693. Springer, 2009.

- [3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [4] Mihir Bellare, Alexandra Boldyreva, and Adam O’Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology–CRYPTO 2007*, pages 535–552. Springer, 2007.
- [5] Michael A Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P Spillane, and Erez Zadok. Don’t thrash: How to cache your hash on flash. *Proceedings of the VLDB Endowment*, 5(11):1627–1637, 2012.
- [6] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [7] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. Order-preserving symmetric encryption. In *Advances in Cryptology–EUROCRYPT 2009*, pages 224–241. Springer, 2009.
- [8] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373, 2013.
- [9] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 79–88, 2006.
- [10] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than Bloom. In *Proceedings of the 10th ACM International Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.
- [11] Michael L Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.
- [12] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.
- [13] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
- [14] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.
- [15] Ari Juels and Thomas Ristenpart. Honey encryption: Security beyond the brute-force bound. In *Advances in Cryptology – EUROCRYPT 2014*, pages 293–310, 2014.
- [16] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, pages 644–655, 2015.
- [17] Claude E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

- [18] Gustavus J Simmons. Symmetric and asymmetric encryption. *ACM Computing Surveys*, 11 (4):305–330, 1979.
- [19] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 44–55, 2000.
- [20] Alexander Towell. Bernoulli sets and maps: A probabilistic framework for approximate data structures, 2026. Manuscript in preparation. See https://github.com/queelius/bernoulli_sets.
- [21] Alexander Towell. Algebraic cipher types: A functorial framework for structured computation over trapdoor encodings, 2026. Manuscript in preparation.
- [22] Andrew C Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.