Algebraic Cipher Types: Confidentiality Trade-offs in Type Constructors over Trapdoor Computing

Alexander Towell
lex@metafunctor.com

March 2026

Abstract

A cipher map is a total function on bit strings that implements a trapdoor approximation of a latent function: the untrusted machine evaluates it blindly; only the trusted machine, holding the secret, can encode inputs and decode outputs. We study what happens when algebraic type constructors—products, sums, and exponentials—are applied to cipher values. Each constructor forces a specific confidentiality/functionality trade-off that is absent in plaintext programming. Our main result is a sum-type impossibility: for $A + B$, either the tag (which variant) is hidden and pattern matching requires the trapdoor, or pattern matching is available to the untrusted machine and the tag is leaked. No encoding achieves both simultaneously. We introduce the *orbit closure* of a cipher value under a set of available operations and prove that confidentiality is bounded by the fraction of the cipher space reachable from any single value. A typed-composition discipline turns this bound into a design-time budget: the type of a cipher program determines how deep its orbit can go. We then present two realizations of cipher programs, an expression-tree decomposition with cipher-map cut points (the practical path, exposed via a `@cipher_node` decorator) and a cipher Turing machine (space $O(|Q|\cdot|\Gamma|)$, leaks head movement), and evaluate the cipher Boolean algebra specialization on a 20 Newsgroups Boolean search task.

1 Introduction

Trapdoor computing is a paradigm for evaluating functions on data whose meaning is hidden behind a one-way trapdoor (a cryptographic hash). A trusted machine encodes plaintext values into cipher values—opaque bit strings—and hands them, together with cipher maps (total functions on bit strings), to an untrusted machine. The untrusted machine evaluates the cipher maps and returns results. Without the trapdoor (the secret seed), it cannot decode the results or determine what function it computed. The formal framework of cipher maps, their four defining properties (totality, representation uniformity, correctness, composability), and three concrete constructions (HashSet, entropy map, trapdoor boolean algebra) are developed in [18].

Ordinary programming languages build complex types from simpler ones using a small set of type constructors: products ($A \times B$, pairs and records), sums ($A + B$, tagged unions and variants), and exponentials ($A \to B$, functions). In plaintext computing, these constructors are transparent: a product exposes both components, a sum carries a tag that enables pattern matching, and a function is called by providing an argument. When values are hidden behind a trapdoor, each of these operations introduces a confidentiality cost that has no plaintext analogue.

This paper identifies and formalizes those costs. The main contributions are:

1. **Sum-type impossibility** (Theorem 4.2). For cipher values of sum type $A+B$, tag hiding and pattern matching by the untrusted machine are mutually exclusive. This is an information-theoretic result, independent of the cipher map construction.

2. **Orbit closure and confidentiality bound** (Section 5). We define the orbit closure of a cipher value under available operations, prove monotonicity (Theorem 5.1), and derive an upper bound on confidentiality in terms of orbit size (Theorem 5.3). A typed-composition discipline (Section 5.5) converts this bound into a design-time budget: the type of a cipher program bounds its orbit depth and hence its worst-case confidentiality loss.

3. **Cipher program realizations** (Section 6). We present two concrete ways to decompose a cipher program into cipher-map pieces. Expression-tree decomposition with cipher-node annotations is the practical path; a cipher Turing machine is a second realization, with space $O(|Q| \cdot |\Gamma|)$ at the cost of leaking head movement. The two realizations share a cut-point structure and differ in what residual structure the untrusted machine observes.

We deliberately avoid game-based or simulation-based cryptographic definitions. Privacy in trapdoor computing comes from the one-way hash and the representation uniformity of cipher values, not from access-pattern indistinguishability. This is not ORAM [8], FHE [7], or garbled circuits [21]; it is a distinct paradigm where totality and noise closure provide the confidentiality guarantees.

2 Related Work

Information flow type systems. The sum-type impossibility (Theorem 4.2) is closely related to the implicit flow problem in language-based information security [13]: a conditional branch on a secret value leaks information through the control flow path taken. Our result formalizes this for cipher values: pattern matching on a cipher sum type is an implicit flow that leaks the tag. The difference is that information flow type systems prevent the leak at compile time, while cipher types quantify it at the representation level.

Functional encryption. Functional encryption [3] allows computing $f(x)$ on an encrypted x by issuing function-specific decryption keys. Cipher maps differ in two ways: (1) they are information-theoretic (parameterized by η, ε, δ) rather than game-based, and (2) they are approximate (the untrusted machine evaluates a total function with bounded error, not an exact decryption). The type-constructor trade-offs we identify (tag leakage for sums, correlation leakage for products) apply equally to functional encryption schemes that support algebraic operations.

Searchable symmetric encryption. SSE originates with Song, Wagner, and Perrig [16] and is placed on rigorous foundations by Curtmola et al. [5]. The Boolean search construction in Section 7 provides an alternative with information-theoretic parameters: cipher sets with cipher Boolean AND/OR/NOT, rather than encrypted indices with leakage profiles. Subsequent work established that leakage profiles in SSE are attackable in practice: Islam, Kuzu, and Kantarcioglu [10] demonstrated access-pattern disclosure attacks; Cash et al. [4] and Grubbs, Ristenpart, and Shmatikov [9] mounted leakage-abuse attacks against deployed SSE and encrypted databases; Naveed, Kamara, and Wright [12] did the same for property-preserving encryption. Cipher maps avoid property preservation entirely and limit the adversary to the orbit-closure bound of Theorem 5.3.

Quantitative information flow. The entropy-form confidentiality bound of Theorem 5.3 places this work in the quantitative information flow tradition [15], where leakage is measured as conditional entropy or min-entropy of the secret given the adversary's observations. Our contribution is to specialize this framework to cipher values under an algebraic operation set, yielding a construction-independent bound in terms of orbit size.

Related concepts. Cipher maps realize the oblivious evaluation of a computable function (after Turing [20]) through a total function on bit strings, and the cipher Boolean construction (Section 7) is structurally related to the Bloom filter [2], which it generalizes by allocating an explicit False region alongside the True region. Our experiments sit on the 20 Newsgroups corpus of Lang [11]. Maximizing-confidentiality analyses complementary to this paper's information-theoretic bound appear in a companion work [19].

3 Preliminaries

We recall the cipher map abstraction from [18]. All notation follows that paper; we cite rather than re-derive.

Definition 3.1 (Cipher map [18, Def. 1.1])**.** A *cipher map* for a latent function $f : X \to Y$ is a tuple $(\hat{f}, \mathrm{enc}, \mathrm{dec}, s)$ where $\hat{f} : \{0,1\}^n \to \{0,1\}^n$ is a total function on n-bit strings, $\mathrm{enc} : X \times \{0, \ldots, K-1\} \to \{0,1\}^n$ is an encoding function, $\mathrm{dec} : \{0,1\}^n \to Y \cup \{\perp\}$ is a decoding function, and s is a secret (the trapdoor). The untrusted machine holds $\hat{f}$. The trusted machine holds enc, dec, and s.

A cipher map satisfies four properties, parameterized by $(\eta, \varepsilon, \mu, \delta)$ [18, Sec. 4]:

1. **Totality.** $\hat{f}$ is defined on all 2^n inputs. Out-of-domain outputs are indistinguishable from uniform under the random oracle model [1].
2. **Representation uniformity** (δ-bounded). The marginal distribution of cipher values is δ-close (in total variation distance) to uniform over $\{0,1\}^n$.
3. **Correctness** (η-bounded). For a random in-domain element x and random representation index k, $\Pr[\mathrm{dec}(\hat{f}(\mathrm{enc}(x,k))) \neq f(x)] \leq \eta$.
4. **Composability.** For cipher maps $\hat{f}$ (correctness η_f) and $\hat{g}$ (correctness η_g), the composition $\hat{g} \circ \hat{f}$ has correctness $\eta_{g \circ f} = 1 - (1 - \eta_f)(1 - \eta_g)$.

The Bernoulli error model [17] provides the quantitative foundation: element-wise independence of errors reduces the composition analysis from exponentially many joint failure modes to per-element Bernoulli parameters. We use $h : \{0,1\}^* \to \{0,1\}^n$ throughout to denote a cryptographic hash modeled as a random oracle.

A cipher value $\mathrm{enc}(v, k)$ is a cipher map for the constant function $f_v(x) = v$; we use "cipher value" and "cipher map from a trivial domain" interchangeably.

Definition 3.2 (Trusted and untrusted machines [18, Sec. 5])**.** The *trusted machine* T holds the trapdoor s, enc, dec, and can encode, decode, and inject filler queries. The *untrusted machine* U holds $\hat{f}$ and a collection of cipher values. U can evaluate $\hat{f}$ on any bit string; it cannot decode, distinguish real from filler queries, or determine the domain X.

4 Cipher Type Constructors

We now examine how the standard algebraic type constructors interact with cipher values. For each constructor, there is a choice: apply the cipher operation to the entire constructed type, or construct from already-ciphered components. These two strategies yield different confidentiality and functionality properties. We write $\mathsf{C}(X)$ for the type of cipher values encoding elements of X.

4.1 Void and Unit

The void type $\mathbf{0}$ has no values. A cipher map $\hat{f}$ for a function with domain $\mathbf{0}$ is vacuously correct ($\eta = 0$) and carries no information—it is a total function on $\{0,1\}^n$ that is never queried on a valid encoding. From the untrusted machine's perspective, $\hat{f}$ is indistinguishable from a random function.

The unit type $\mathbf{1}$ has exactly one value, $\star$. A cipher value $\mathrm{enc}(\star, k) \in \{0,1\}^n$ encodes $\star$; with $K(\star) \geq 1$ representations, all cipher values are representations of the same underlying value. The untrusted machine knows that every valid cipher value encodes $\star$ but cannot distinguish valid encodings from noise (by totality). A cipher map $\hat{f} : \{0,1\}^n \to \{0,1\}^n$ for a function from $\mathbf{1}$ is a cipher constant: it maps every valid encoding of $\star$ to a cipher encoding of $f(\star)$.

4.2 Product Types

The product type $A \times B$ consists of pairs (a, b) with $a \in A$ and $b \in B$. In plaintext computing, a product comes with two projections $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$. Under cipher encoding, we have two choices.

Joint encoding: $\mathsf{C}(A \times B)$. Encode the pair (a, b) as a single cipher value $\mathrm{enc}((a, b), k) \in \{0,1\}^n$. The untrusted machine sees one opaque bit string. It cannot determine a or b individually, nor can it detect correlations between a and b. But the projections π_1, π_2 are not available: to extract a from a cipher encoding of (a, b), one would need a cipher map $\hat{\pi}_1 : \{0,1\}^n \to \{0,1\}^n$ such that $\mathrm{dec}_A(\hat{\pi}_1(\mathrm{enc}((a, b), k))) = a$—which requires a separate cipher map construction for each projection.

Component-wise encoding: $\mathsf{C}(A) \times \mathsf{C}(B)$. Encode a and b independently as $(\mathrm{enc}_A(a, k_1), \mathrm{enc}_B(b, k_2))$. The untrusted machine sees a pair of cipher values. Projections are trivially available (just take the first or second component). But the joint distribution of (a, b) is no longer hidden: even if each component has marginal representation uniformity ($\delta \approx 0$), the correlation between a and b is preserved in the joint distribution of the cipher pair. This is the encoding granularity principle of [18, Sec. 9]: coarser encoding hides more correlations but blocks more operations.

Proposition 4.1 (Product confidentiality trade-off)**.** *Let A and B be finite types.*

1. *Under joint encoding $\mathsf{C}(A \times B)$, the untrusted machine cannot distinguish (a_1, b_1) from (a_2, b_2) (up to δ), but projections require a cipher map (and thus a separate construction by the trusted machine).*

2. *Under component-wise encoding $\mathsf{C}(A) \times \mathsf{C}(B)$, projections are free, but an adversary observing N pairs $(\mathrm{enc}_A(a_i, k_{1i}), \mathrm{enc}_B(b_i, k_{2i}))$ can estimate the joint distribution of (A, B) to arbitrary precision as $N \to \infty$.*

Proof. Part (1): Under joint encoding with representation uniformity δ, each cipher value is δ-close to uniform over $\{0,1\}^n$. Without the trapdoor, the untrusted machine cannot decode. Projections

require mapping a cipher encoding of (a, b) to a cipher encoding of a; this is a function from the code space of $A \times B$ to the code space of A, which is exactly a cipher map and requires construction by the trusted machine.

Part (2): Under component-wise encoding, the i-th observation is $(\mathrm{enc}_A(a_i, k_{1i}), \mathrm{enc}_B(b_i, k_{2i}))$. Even with perfect marginal uniformity ($\delta = 0$), each enc_A is a deterministic function of (a_i, k_{1i}). If the same representation index is reused, repeated values produce identical cipher values; even without reuse, the number of distinct cipher values associated with each latent value is finite ($K(a)$ for each a). Given N observations, the adversary builds the empirical joint distribution over cipher-value pairs. Since each cipher value maps to at most one latent value (given a fixed representation index), the empirical joint distribution of cipher pairs converges to a mixture of the latent joint distribution. By standard convergence results, $N = O(|A| \cdot |B|/\xi^2)$ observations suffice to estimate the joint to accuracy ξ in total variation distance. □

4.3 Sum Types

The sum type $A + B$ is a tagged union: each value is either $\mathrm{inl}(a)$ for some $a \in A$, or $\mathrm{inr}(b)$ for some $b \in B$. The tag (left vs. right) enables pattern matching: given a value of type $A + B$, one inspects the tag to determine which handler to apply.

In plaintext computing, the tag is always available and pattern matching is free. Under cipher encoding, this is no longer the case.

Theorem 4.2 (Sum-type impossibility)**.** *Let A and B be non-empty finite types with $|A| \geq 1$ and $|B| \geq 1$. For cipher values of sum type $A + B$, the following two properties are mutually exclusive:*

1. ***Tag hiding**: the untrusted machine cannot determine, from a cipher value alone, whether it encodes an A-value or a B-value with advantage better than δ (the representation uniformity parameter).*

2. ***Untrusted pattern matching**: the untrusted machine can route a cipher value to the correct handler (A-handler or B-handler) without assistance from the trusted machine.*

Proof. We prove the two directions by exhibiting the information-theoretic constraint.

Joint encoding ($\mathsf{C}(A + B)$) achieves tag hiding but blocks pattern matching. Under joint encoding, each value $v \in A + B$ (whether $\mathrm{inl}(a)$ or $\mathrm{inr}(b)$) is encoded as a single cipher value $\mathrm{enc}(v, k) \in \{0,1\}^n$. Representation uniformity (δ-bounded) guarantees that the marginal distribution of $\mathrm{enc}(v, k)$ over random v and k is δ-close to uniform on $\{0,1\}^n$. In particular, the set of cipher values encoding A-values and the set encoding B-values are not efficiently distinguishable by the untrusted machine: both are subsets of $\{0,1\}^n$, and their union is δ-close to covering $\{0,1\}^n$ uniformly. Therefore the tag (which variant) is hidden up to δ.

Now suppose the untrusted machine could nonetheless perform pattern matching: given $c = \mathrm{enc}(v, k)$, it could determine whether to apply the A-handler or the B-handler. This requires a function $\tau : \{0,1\}^n \to \{A, B\}$ such that $\tau(\mathrm{enc}(\mathrm{inl}(a), k)) = A$ for all a, k and $\tau(\mathrm{enc}(\mathrm{inr}(b), k)) = B$ for all b, k. But τ is exactly a distinguisher between A-encodings and B-encodings. Such a distinguisher exists only if the two sets of cipher values are disjoint (or nearly so) in a way detectable without the trapdoor. If τ succeeds with probability $1 - \gamma$, then the tag is leaked with advantage at least $1 - \gamma - \delta$, contradicting tag hiding when γ is small.

Component-wise encoding ($\mathsf{C}(A) + \mathsf{C}(B)$) permits pattern matching but leaks the tag. Under component-wise encoding, the value is stored as a tagged pair: either $(\mathtt{L}, \mathrm{enc}_A(a, k))$ or

$(\mathtt{R}, \mathrm{enc}_B(b, k))$, where the tag `L` or `R` is in the clear (or distinguishable by the encoding structure—for instance, A-cipher-values and B-cipher-values occupy disjoint regions of bit-string space, or carry a type identifier). The untrusted machine reads the tag and routes to the correct handler. Pattern matching is available.

But the tag is observable. Given a cipher value, the untrusted machine learns whether it encodes an A-value or a B-value. For any distribution D on $A + B$, the fraction of queries that are A-values is observable, leaking $\Pr_{v \sim D}[v \in A]$.

No intermediate encoding achieves both. Suppose an encoding achieves both tag hiding (advantage $\leq \delta'$ for some small δ') and untrusted pattern matching (success probability $\geq 1 - \gamma$). Pattern matching requires a function $\tau : \{0,1\}^n \to \{A, B\}$ that is correct on valid encodings with probability $\geq 1 - \gamma$. Tag hiding requires that no such function exists with advantage better than δ'. Let $p = \Pr[v \in A]$ be the prior probability of an A-value. A trivial strategy that ignores the encoding and always guesses the majority type achieves accuracy $\max(p, 1-p)$. The advantage of τ over this baseline is at least $(1-\gamma) - \max(p, 1-p)$. Tag hiding requires this advantage to be at most δ'. Therefore $(1-\gamma) - \max(p, 1-p) \leq \delta'$, which means either $\gamma \geq 1 - \max(p, 1-p) - \delta'$ (pattern matching fails often) or $\delta' \geq (1-\gamma) - \max(p, 1-p)$ (the tag is substantially leaked). For the balanced case $p = 1/2$, this simplifies to $1/2 - \gamma \leq \delta'$. Unbalanced distributions ($p \neq 1/2$) make the impossibility stronger: the baseline accuracy is higher, leaving less room for the encoding to add information without leaking the tag. □

Remark 4.1 (Contrast with products). Products leak correlations; sums leak the tag. The asymmetry is structural. For a product $A \times B$, the two components are always both present; the question is whether their relationship (correlation) is visible. For a sum $A + B$, only one component is present; the question is whether which one is visible. A product hides *what values co-occur*; a sum hides *what type is occupied*.

Remark 4.2 (Trusted pattern matching). Under joint encoding $\mathsf{C}(A + B)$, pattern matching is still possible—by the trusted machine. The trusted machine decodes $\mathrm{dec}(c)$, inspects the tag, applies the appropriate handler in plaintext, and re-encodes the result. This requires a round trip through the trusted machine for every case analysis. The cost is a communication round, not an impossibility. The impossibility is for *untrusted* pattern matching without the trapdoor.

Example 4.1 (Cipher optionals). The optional type $\mathrm{Maybe}(A) = \mathbf{1} + A$ is a sum. Under $\mathsf{C}(\mathbf{1} + A)$, the untrusted machine cannot tell whether a cipher value encodes Nothing or $\mathrm{Just}(a)$. Under $\mathsf{C}(\mathbf{1}) + \mathsf{C}(A)$, the untrusted machine knows whether the value is present, but not what it is. For a database application, the first encoding hides which records have null fields; the second reveals the null pattern but permits null-checking without the trapdoor.

4.4 Exponential Types

The exponential type $B^A = A \to B$ is the type of functions from A to B. A cipher map $\hat{f}$ for $f : A \to B$ is already an element of the cipher exponential type: it is a total function on $\{0,1\}^n$ that, when restricted to valid encodings of A, produces (approximate) encodings of B-values.

This observation, developed in [18], means the cipher exponential type is not a new construction but the cipher map abstraction itself. The four properties (totality, representation uniformity, correctness, composability) are exactly the conditions that make $\hat{f}$ a well-behaved element of $\mathsf{C}(A \to B)$.

The encoding granularity trade-off from Section 4.2 extends to exponentials. Given cipher maps $\hat{f} : \{0,1\}^n \to \{0,1\}^n$ and $\hat{g} : \{0,1\}^n \to \{0,1\}^n$, applying them independently (component-wise)

to the two components of a product cipher value reveals that both functions are being applied—leaking the computation pattern. Encoding the pair of function applications as a single cipher map $\widehat{(f,g)} : \{0,1\}^n \to \{0,1\}^n$ for the product function $(f,g)(a,b) = (f(a), g(b))$ hides the computation pattern but requires a dedicated construction.

5 Orbit Closure and Information Leakage

The type constructors of Section 4 describe static confidentiality properties: what the untrusted machine can learn from the structure of a single cipher value. We now develop a tool for analyzing dynamic confidentiality: what the untrusted machine can learn by actively applying the cipher maps it holds.

5.1 Definition

Definition 5.1 (Orbit closure)**.** Let $F = \{\hat{f}_1, \ldots, \hat{f}_k\}$ be a set of cipher maps (total functions $\{0,1\}^n \to \{0,1\}^n$) available to the untrusted machine, and let $c \in \{0,1\}^n$ be a cipher value. The *orbit closure* of c under F is the smallest set $\mathrm{orbit}_F(c) \subseteq \{0,1\}^n$ such that:

1. $c \in \mathrm{orbit}_F(c)$, and
2. if $c' \in \mathrm{orbit}_F(c)$ and $\hat{f}_i \in F$, then $\hat{f}_i(c') \in \mathrm{orbit}_F(c)$.

Equivalently, $\mathrm{orbit}_F(c)$ is the set of all bit strings reachable from c by composing operations in F in any order and any number of times.

Since $\{0,1\}^n$ is finite, $\mathrm{orbit}_F(c)$ is always finite and can be computed by iterative expansion (breadth-first search from c, applying each $\hat{f}_i$ at each step, until no new values appear).

Remark 5.1. The orbit closure depends only on the cipher maps F available to the untrusted machine and the starting cipher value c. It does not depend on the latent function, the domain, or the trapdoor. This makes it a tool for analysis by the trusted machine designer: given the operations you plan to expose, how much can the untrusted machine explore?

5.2 Monotonicity

Theorem 5.1 (Monotonicity of orbit closure)**.** *If $F \subseteq G$ (every operation in F is also in G), then $\mathrm{orbit}_F(c) \subseteq \mathrm{orbit}_G(c)$ for all $c \in \{0,1\}^n$.*

Proof. By induction on the iterative construction of $\mathrm{orbit}_F(c)$.

Base case. $c \in \mathrm{orbit}_F(c)$ and $c \in \mathrm{orbit}_G(c)$ by definition.

Inductive step. Suppose $c' \in \mathrm{orbit}_F(c)$ and $c' \in \mathrm{orbit}_G(c)$ (inductive hypothesis). For any $\hat{f}_i \in F$, we have $\hat{f}_i(c') \in \mathrm{orbit}_F(c)$ by Definition 5.1. Since $F \subseteq G$, $\hat{f}_i \in G$, so $\hat{f}_i(c') \in \mathrm{orbit}_G(c)$ by Definition 5.1 applied to G.

Every element of $\mathrm{orbit}_F(c)$ is therefore also in $\mathrm{orbit}_G(c)$. □

The interpretation is direct: exposing more operations to the untrusted machine can only enlarge (never shrink) the set of cipher values it can reach. Each additional operation is an additional edge in the reachability graph over $\{0,1\}^n$.

Corollary 5.2. $\mathrm{orbit}_\emptyset(c) = \{c\}$ *for all c. If the untrusted machine has no cipher maps, it learns nothing beyond the cipher value it was given.*

Proof. With $F = \emptyset$, only the base case of the orbit definition applies. □

5.3 Confidentiality Bound

The orbit closure quantifies reachability. We now connect it to confidentiality via an information-theoretic bound: each distinct cipher value in $\mathrm{orbit}_F(c)$ is a signal the adversary observes, and the number of such signals upper-bounds how much can be learned about the latent value.

Let X denote the random latent value encoded by c (drawn from the distribution over inputs that the trusted machine uses) and let $\mathcal{V}_F(c) = \mathrm{orbit}_F(c)$ denote the adversary's view: the set of cipher values reachable from c under F. The adversary's residual uncertainty about X is quantified by the conditional entropy $H(X \mid \mathcal{V}_F(c))$.

Theorem 5.3 (Confidentiality bound, entropy form)**.** *Let F be a set of cipher maps available to the untrusted machine and c a cipher value. Then*

$$H(X \mid \mathcal{V}_F(c)) \;\geq\; H(X) - \log_2 |\mathrm{orbit}_F(c)|.$$

Proof. The view $\mathcal{V}_F(c)$ takes at most $|\mathrm{orbit}_F(c)|$ distinct values as X ranges over its distribution, since the orbit is a finite set of bit strings computable from c. Therefore

$$I(X; \mathcal{V}_F(c)) \;\leq\; H(\mathcal{V}_F(c)) \;\leq\; \log_2 |\mathrm{orbit}_F(c)|,$$

and $H(X \mid \mathcal{V}_F(c)) = H(X) - I(X; \mathcal{V}_F(c))$. □

Corollary 5.4 (Confidentiality bound, set form)**.** *Let $S_F(c) \subseteq X$ denote the set of latent values the adversary cannot rule out from $\mathcal{V}_F(c)$, and define the* confidentiality *of c under F as* $\mathrm{conf}_F(c) = 1 - |S_F(c)|/|X|$. *Then*

$$\mathrm{conf}_F(c) \;\geq\; 1 - \frac{|\mathrm{orbit}_F(c)|}{|X|}.$$

Proof. Each element of $S_F(c)$ is distinguished from the others by at least one observable feature of the view, and the view takes at most $|\mathrm{orbit}_F(c)|$ distinct values, so $|S_F(c)| \leq |\mathrm{orbit}_F(c)|$. The bound follows. □

Remark 5.2 (Loose bound, tight in the worst case). Both forms are loose: they ignore the functional relationships between orbit elements (the adversary knows $c' = \hat{f}(c)$, which constrains the possible latent values beyond mere set size) and the algebraic structure of F. Tighter bounds for specific operations are an open problem. Equality is approached when orbit elements behave like independent signals about X, e.g., when the cipher maps in F are modeled as random oracles [1].

Remark 5.3 (Extremes). When $|\mathrm{orbit}_F(c)| \geq |X|$, the set-form bound is vacuous; in principle the orbit can identify every latent value. When $|\mathrm{orbit}_F(c)| = 1$ (no operations available, per Corollary 5.2), confidentiality is maximal: $H(X \mid \mathcal{V}_F(c)) = H(X)$ and $\mathrm{conf}_F(c) = 1 - 1/|X|$.

5.4 Examples

Example 5.1 (Boolean operations). Let $X = \{0, 1\}$ (Boolean) with cipher maps $\widehat{\mathrm{AND}}$ and $\widehat{\mathrm{NOT}}$ available to the untrusted machine. Given a cipher value c encoding some unknown Boolean x, the untrusted machine can compute:

$$c_1 = \widehat{\mathrm{NOT}}(c),$$
$$c_2 = \widehat{\mathrm{AND}}(c, c_1) = \widehat{\mathrm{AND}}(c, \widehat{\mathrm{NOT}}(c)).$$

Now, $\mathrm{AND}(x, \mathrm{NOT}(x)) = 0$ for all x (up to correctness η). So c_2 is an (approximate) cipher encoding of the constant FALSE. If the untrusted machine has a reference cipher value c_{F} known to encode FALSE (for instance, from a prior computation $\widehat{\mathrm{AND}}(c', \widehat{\mathrm{NOT}}(c'))$ for some other cipher value c'), it can compare c_2 with c_{F}.

This comparison reveals information: if $c_2 = c_{\mathrm{F}}$ (same representation), the untrusted machine confirms the tautology and potentially identifies the representation index. Writing $F_0 = \{\widehat{\mathrm{AND}}, \widehat{\mathrm{NOT}}\}$, the orbit $\mathrm{orbit}_{F_0}(c)$ includes c, c_1, c_2, and all further compositions—quickly covering a nontrivial fraction of the cipher space for small n.

Example 5.2 (Successor on a bounded type). Let $X = \{0, 1, \ldots, m-1\}$ (integers mod m) with a single cipher map $\widehat{\mathrm{succ}}$ implementing the successor function $x \mapsto (x+1) \bmod m$. Given any cipher value c encoding some x:

$$\mathrm{orbit}_{\{\widehat{\mathrm{succ}}\}}(c) = \{c,\ \widehat{\mathrm{succ}}(c),\ \widehat{\mathrm{succ}}^2(c),\ \ldots,\ \widehat{\mathrm{succ}}^{m-1}(c)\}.$$

After m applications, the orbit returns to c (since successor is cyclic on $\mathbb{Z}/m\mathbb{Z}$). The orbit has exactly m elements (assuming $\eta = 0$). The untrusted machine does not learn which element is $\mathrm{enc}(0, k)$, but it learns the cycle structure: there exists a cycle of length m starting from c. This reveals $m = |X|$, the domain cardinality.

By Theorem 5.3, the confidentiality is $1 - m/2^n$. For $m \ll 2^n$, this is close to 1: the orbit covers a tiny fraction of the cipher space. For m close to 2^n, confidentiality degrades.

Example 5.3 (Branching). Suppose the untrusted machine holds cipher maps $\hat{f}$ and $\hat{g}$ and evaluates both on a cipher value c, obtaining $\hat{f}(c)$ and $\hat{g}(c)$. The orbit of c under $\{\hat{f}, \hat{g}\}$ includes both results plus all further compositions. If $\hat{f}$ and $\hat{g}$ are the two branches of a conditional (the "then" and "else" handlers), both cipher results are visible to the untrusted machine—even though in plaintext computing, only one branch would be taken.

This is a form of information leakage specific to trapdoor computing: the untrusted machine does not know which branch is "correct," but it sees both outcomes, enlarging the orbit and reducing confidentiality. Mitigating this requires either (a) evaluating both branches and having the trusted machine select the correct result, or (b) encoding the branch selection as part of the cipher map (a joint encoding of the conditional). Strategy (a) doubles the computation; strategy (b) requires a dedicated cipher map construction.

Remark 5.4 (Active probing via Boolean operations). Exposing cipher Boolean operations creates an active probing risk beyond passive observation. Given cipher AND and a cipher value c, the adversary computes $\mathrm{AND}(c, c)$, $\mathrm{AND}(c, c')$ for other values c', and checks structural consistency. Over many probes, the adversary partitions cipher values into equivalence classes corresponding to latent Boolean values. Each probe enlarges the orbit, and confidentiality degrades per Theorem 5.3. This is a concrete instance of orbit closure: Boolean operations are probing tools that trade functionality for confidentiality.

5.5 Typed Composition Chains

The active probing risk motivates controlling the orbit by construction. The orbit grows because the adversary can feed cipher map outputs back as inputs, composing operations to reach new cipher values. A direct mitigation: *prevent self-composition by typing the cipher spaces.*

Definition 5.2 (Typed composition chain)**.** A *typed composition chain* of depth k is a sequence of cipher maps $\hat{f}_0, \ldots, \hat{f}_{k-1}$ where $\hat{f}_i : \mathsf{C}(A)_i^{a_i} \to \mathsf{C}(A)_{i+1}$ has arity $a_i \geq 1$ over the cipher space $\mathsf{C}(A)_i$, and the cipher spaces $\mathsf{C}(A)_0, \ldots, \mathsf{C}(A)_k$ are *distinct* (disjoint hash value sets, different secrets).

Since $\mathsf{C}(A)_i \neq \mathsf{C}(A)_j$ for $i \neq j$, the output of $\hat{f}_i$ cannot be fed into $\hat{f}_j$ for $j \neq i+1$. The adversary can compose at most k operations in sequence and cannot loop.

Proposition 5.5 (Orbit bound for typed chains)**.** *Let $V_0 \subseteq \mathsf{C}(A)_0$ be a set of m initial cipher values available to the adversary, and let $F = \{\hat{f}_0, \ldots, \hat{f}_{k-1}\}$ be a typed composition chain of depth k with arities $a_0, \ldots, a_{k-1}$. Define $N_0 = m$ and $N_{i+1} = N_i^{a_i}$. Then*

$$|\mathrm{orbit}_F(V_0)| \;\leq\; \sum_{i=0}^{k} N_i,$$

a finite bound determined entirely by $(k, m, a_0, \ldots, a_{k-1})$ and independent of the cipher-space sizes $|\mathsf{C}(A)_i|$. Two useful special cases:

- ***Single-value start*** *($m = 1$), any arities: $|\mathrm{orbit}_F(V_0)| \leq 1 + k$.*
- ***Unary chain*** *($a_i = 1$), any m: $|\mathrm{orbit}_F(V_0)| \leq m(k+1)$.*

The corresponding confidentiality bound follows from Corollary 5.4.

Proof. Partition $\mathrm{orbit}_F(V_0)$ by level: $L_0 = V_0 \subseteq \mathsf{C}(A)_0$, and $L_{i+1} \subseteq \mathsf{C}(A)_{i+1}$ is the image of $\hat{f}_i$ applied to all a_i-tuples of elements of L_i. Distinct cipher spaces prevent back-references, so every orbit element lies in exactly one L_i. Since $|L_i| \leq |L_{i-1}|^{a_{i-1}}$, induction gives $|L_i| \leq N_i$, and therefore $|\mathrm{orbit}_F(V_0)| = \sum_i |L_i| \leq \sum_i N_i$. For $m = 1$, $N_i = 1$ at every level (identical inputs produce one output for a deterministic cipher map), so the sum is $1 + k$. For $a_i = 1$, $N_i = m$ at every level, so the sum is $m(k+1)$. □

Example 5.4 (Depth-limited Boolean search). Consider Boolean search with maximum query depth 2 (e.g., $w_1 \wedge w_2$ but not $w_1 \wedge w_2 \wedge w_3$):

- Cipher sets output to $\mathsf{C}(\mathrm{Bool})_0$.
- $\mathrm{AND}_0 : \mathsf{C}(\mathrm{Bool})_0 \times \mathsf{C}(\mathrm{Bool})_0 \to \mathsf{C}(\mathrm{Bool})_1$ (binary, $a_0 = 2$).
- $\mathrm{AND}_1 : \mathsf{C}(\mathrm{Bool})_1 \times \mathsf{C}(\mathrm{Bool})_1 \to \mathsf{C}(\mathrm{Bool})_2$ (binary, $a_1 = 2$).
- No AND_2 is defined.

Starting from a single cipher Boolean c (one membership-query output), the orbit has size at most $1 + k = 3$: namely $\{c, \mathrm{AND}_0(c,c), \mathrm{AND}_1(\mathrm{AND}_0(c,c), \mathrm{AND}_0(c,c))\}$. With m membership-query outputs as initial values, the bound is $m + m^2 + m^4$, still finite and controlled by the chain depth. Contrast with a single shared cipher Boolean space, where the adversary can self-compose indefinitely and the orbit grows up to 2^n.

The general principle: **the type system controls the orbit**. By choosing how many cipher space levels to define, the system designer sets the maximum composition depth and therefore the maximum orbit size and minimum confidentiality. This is a compile-time (construction-time) decision with no runtime cost: the untrusted machine simply does not have cipher maps for deeper levels.

6 Realizing Cipher Programs

The type-theoretic results of Sections 4 and 5 bound what any cipher-program realization can achieve. We now examine two concrete realizations, each decomposing a program into cipher-map pieces along a different axis. A cipher Turing machine treats the program as a repeated transition step; expression-tree decomposition treats the program as a DAG whose selected subtrees are replaced by cipher maps. The two realizations leak different residual structure and occupy different points in the space-time-confidentiality trade-off. We present them in turn and then make explicit the common cut-point pattern they share (Section 6.3).

6.1 Cipher Turing Machines

A cipher map for $f : X \to Y$ is a lookup table of size $O(|X|)$, which is prohibitive when $|X|$ is large (e.g., string processing with $|X| = |\Sigma|^{\ell}$). A Turing machine is an alternative whose space depends on the *program*, not the *data*.

Definition 6.1 (Cipher Turing machine)**.** A *cipher Turing machine* consists of a finite state set Q, a tape alphabet Γ with blank $\sqcup$, an initial state q_0, a halting state q_{halt}, and a cipher map $\hat{\delta}_T$ for the transition function $\delta_T : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$. Each tape cell is a single cipher value in $\mathsf{C}(Q \times \Gamma)$: the cell stores a cipher encoding of the (state, symbol) pair at that cell, with the distinguished state $\varnothing \in Q$ marking cells without the head. On entry, the trusted machine encodes the head position by placing a cipher value for $(q_0, \sqcup)$ (or the first input symbol) at the start cell and cipher encodings of $(\varnothing, \sqcup)$ elsewhere. At each step the untrusted machine applies $\hat{\delta}_T$ to the current-head cell, obtaining a pair (c', d) where c' is a cipher value in $\mathsf{C}(Q \times \Gamma)$ and $d \in \{L, R, S\}$ is the head direction in the clear. The untrusted machine writes c' back to the current cell, shifts the "head" marker according to d (by swapping the state component of the neighbouring cell with the current cell via a second cipher map for the shift operation), and repeats. The head direction is not cipher-encoded: the untrusted machine must know which cell to access next. Ciphering the direction would require evaluating all three neighbouring cells on every step, tripling per-step work while still revealing that a step occurred.

Proposition 6.1 (Cipher TM space complexity)**.** *The cipher map $\hat{\delta}_T$ has space $O(|Q| \cdot |\Gamma|)$ (plus $-\log_2 \varepsilon$ bits per entry for the noise budget and $H(Q \times \Gamma \times \{L, R, S\})$ bits per entry for value encoding, per [18, Sec. 6]), independent of the input domain size $|X|$. When $|X| \gg |Q| \cdot |\Gamma|$, this is a qualitative improvement over the $O(|X|)$ space of a lookup-table cipher map, at the cost of $O(T(|x|))$ steps per evaluation instead of $O(1)$, where T is the running time of the plaintext TM.*

Proposition 6.2 (Head-movement leakage)**.** *For a deterministic cipher TM with $\eta = 0$, the head-movement sequence $d_1, d_2, \ldots \in \{L, R, S\}^*$ is a deterministic function of the input. Two inputs that produce different head-movement sequences are distinguishable by the untrusted machine without the trapdoor.*

Proof. With $\eta = 0$, $\hat{\delta}_T$ reproduces the plaintext Turing machine's transitions exactly, so the head-movement sequence is identical to the plaintext sequence, which is a deterministic function of the input. Distinct sequences are directly observable by the untrusted machine. □

Remark 6.1 (Comparison with ORAM). Oblivious RAM [8] is designed precisely to hide access patterns. A cipher TM makes no such guarantee: tape contents and the transition function are opaque, but the sequence of cells visited is not. This is weaker than ORAM but requires no $O(\log N)$ per-access overhead. Non-zero η injects noise into the head-movement sequence, but the noise is seed-systematic rather than random, so it is unreliable as an access-pattern obfuscation mechanism.

6.2 Expression-Tree Decomposition

The type-theoretic trade-offs of Sections 4.2 and 4.3 have a direct operational realization: given a program represented as an expression tree, which subtrees should be replaced by cipher maps?

Definition 6.2 (Cipher node annotation)**.** Given a program P represented as an expression tree, a *cipher node annotation* marks a subset of nodes $\{n_1, \ldots, n_k\}$ in the tree. Each marked node n_i, together with its entire subtree, is replaced by a cipher map: the subtree is evaluated at construction time (on the trusted machine) over a specified domain, and the resulting mapping is stored as a cipher map backed by a perfect hash function [6]. Unmarked nodes above the cipher nodes become a *combiner*: itself a cipher map built over the hash-value outputs of the components.

The placement of cipher node annotations controls the encoding granularity:

- **Annotation at the root**: the entire program is one cipher map. Maximum confidentiality (no intermediate values exposed), maximum space ($O(|X|)$ where X is the input domain of the full program).
- **Annotation at the leaves**: each leaf function is a separate cipher map. The combiner chains them. Intermediate cipher values are visible to the untrusted machine, leaking correlations between components (as in the product-type trade-off of Section 4.2).
- **Annotation at intermediate nodes**: a spectrum between the two extremes. Deeper annotations absorb more computation into cipher maps (more confidential, more space); shallower annotations expose more structure (less space, more leakage).

Remark 6.2 (Shared variables and compositional leakage). When two cipher nodes f and g share an input variable x, the untrusted machine observes two cipher values $\mathrm{enc}_f(x, k_f)$ and $\mathrm{enc}_g(x, k_g)$ derived from the same latent x. Even if both encodings are individually δ-close to uniform, the joint distribution of $(\mathrm{enc}_f(x), \mathrm{enc}_g(x))$ preserves the correlation (as in [18, Prop. 9.1]). The only mitigation is to merge f and g into a single cipher node (eliminating the shared intermediate) or to inject noise queries that dilute the correlation signal.

Implementation via tracing. A practical realization uses Python function decorators and runtime tracing. Functions marked with a `@cipher_node` decorator are intercepted during a tracing pass over the input domain. The tracing records which cipher nodes are called, with what arguments, and what they return. From this trace, the system:

1. builds a cipher map for each traced cipher node,
2. builds a combiner cipher map over the hash-output tuples of the components,
3. infers the input routing (which program arguments feed which cipher node) from the traced call patterns.

Short-circuit evaluation (e.g., Python's `x if cond else y`) is handled by computing all component cipher maps for every input, even when the original program skips some branches. The combiner's domain includes hash tuples for both branches. The resulting composed cipher program evaluates entirely on the untrusted machine with no round trips to the trusted machine.

6.3 Cut-Point Structure Common to Both Realizations

Both realizations implement the same abstract move.

Definition 6.3 (Cut point)**.** Given a plaintext computation P represented as a directed graph with plaintext functions at nodes and data flow along edges, a *cut point* is a node whose subgraph of ancestors is replaced, during construction, by a single cipher map. Edges entering the cut point carry plaintext values on the trusted machine and cipher values on the untrusted machine; edges leaving the cut point carry cipher values throughout. A *cut-point set* is a collection of cut points such that every input of the plaintext computation reaches at least one cut point, and the cipher maps at the cut points together with a *combiner* cipher map for the nodes above them compute the full program on the untrusted machine.

Mapping the two realizations onto cut points. A cipher TM places a single cut point at the transition function δ_T, with a self-loop: the cut-point output (c', d) feeds the next step's tape cell, which feeds δ_T again. Expression-tree decomposition places multiple cut points in a finite-depth DAG, one at each `@cipher_node`-annotated node, with the combiner cipher map above them. The number of cut points, their depth in the computation, and the cipher spaces between them are the three design knobs.

What leaks above the cut points. Exactly what remains plaintext above the cut points is the leakage profile. For a cipher TM, this is the head-movement sequence (one clear direction per step) and the halting time, nothing else. For expression-tree decomposition, this is the collection of intermediate cipher values flowing between cipher nodes, which leak correlations whenever multiple cut points share latent inputs.

What the typed-chain bound says in each case. Proposition 5.5 bounds the adversary's orbit by $\sum_i N_i$ where the levels i correspond to the depth of cut points along any computation path. For a cipher TM running for T steps with self-loop at a single (binary-arity, if state and symbol are distinct inputs) cut point, a naive reading gives $N_i = 1$ for a single initial tape configuration so the orbit is bounded by $1 + T$; with m distinct initial tape contents considered as a population, the bound is $m \cdot (T + 1)$. For an expression tree of depth k with binary combinators, the bound is $\sum_{i=0}^{k} m^{2^i}$ where m is the number of distinct latent inputs. Typing the cipher spaces between cut points is what prevents the self-loop in either realization from degenerating into an unbounded orbit.

Example 6.1 (Regex matching as cipher TM vs. expression tree)*.* Consider matching a fixed regular expression R of size r against a string $x \in \Sigma^\ell$. As a cipher TM, Q is the set of DFA states for R (size $O(r)$), $\Gamma = \Sigma \cup \{\sqcup\}$, and $\hat{\delta}_T$ is a single cipher map of space $O(r \cdot |\Sigma|)$, evaluated ℓ times per match. Leakage: the head moves right exactly ℓ times, revealing ℓ. As an expression tree, the program is a fold over the string with ℓ cut points at the transition function; the combiner cipher map above them has domain $\mathsf{C}(Q)^\ell$ so its construction cost is exponential in ℓ. Cipher TM is preferable for long strings; expression-tree decomposition is preferable when ℓ is small and string length must be hidden. Choosing between the two is exactly a cut-point placement problem.

7 Cipher Boolean Algebra and Evaluation

The cipher Boolean type $\mathsf{C}(\mathrm{Bool})$ specializes the sum-type trade-off to the most fundamental algebraic structure and provides the concrete instantiation we evaluate on a real corpus.

7.1 Cipher Boolean Type

Definition 7.1 (Cipher Boolean type)**.** A cipher Boolean type over $\{0,1\}^n$ is a partition of the hash space into three disjoint regions:

- $T \subset \{0,1\}^n$: the True region ($|T|/2^n = p_{\text{true}}$),
- $F \subset \{0,1\}^n$: the False region ($|F|/2^n = p_{\text{false}}$),
- $N = \{0,1\}^n \setminus (T \cup F)$: the noise region.

The partition is derived from a secret via a pseudorandom permutation. Without the secret, membership in T, F, or N is indistinguishable from random.

Remark 7.1 (Generalization beyond Bool)*.* Definition 7.1 generalizes to any finite type Y. A cipher type $\mathsf{C}(Y)$ partitions $\{0,1\}^n$ into $|Y|$ value regions plus a noise region, with $|R_y|/2^n \propto \Pr[Y = y]$ (Shannon-optimal allocation per the source coding theorem [14]). Operations $f : Y_1 \times \cdots \times Y_k \to Z$ become cipher maps over the product of input regions. The cipher Boolean is the special case $Y = \{\text{True}, \text{False}\}$; the same construction applies to enumerated types, bounded integers, or any codomain where the output distribution is known.

7.2 Noise Unreliability

The noise region N plays a structural role beyond mere error tolerance.

Proposition 7.1 (Noise unreliability)**.** *Let* AND*,* OR*,* NOT *be cipher maps implementing Boolean operations over the cipher Boolean type. If any input is a noise value $n \in N$, the output is* uncorrelated *with the intended Boolean result: it lands in T, F, or N according to the allocation ratios p_T, p_F, p_N, independent of the other input's value.*

Proof. The AND, OR, and NOT cipher maps are constructed over the domain $T \cup F$. An input from N is outside this domain. Since the cipher map is total (backed by a PHF with random noise in unused slots), a noise input maps to a random slot whose contents decode to a value in T with probability p_T, in F with probability p_F, and in N with probability p_N. For the standard allocation ($p_T = 0.05$, $p_F = 0.90$, $p_N = 0.05$), a noise input produces False 90% of the time, True 5%, and noise 5%. Crucially, this output is independent of the other (valid) input: the cipher map cannot "correct" for a noise input because it does not know which inputs are noise. □

Remark 7.2 (Noise prevents output certainty)*.* The noise region ensures that an adversary observing a sequence of Boolean cipher map evaluations cannot achieve certainty about any individual result. Consider an adversary that repeatedly ANDs a cipher value c with random cipher values $r_1, r_2, \ldots$. If $c \in T$ (True), then $\text{AND}(c, r_i)$ produces True when $r_i \in T$ and False when $r_i \in F$. But when $r_i \in N$ (5% probability), the output is noise, which the adversary cannot distinguish from a legitimate False. After k ANDs, the adversary has seen some True and some False/noise results, but cannot determine with certainty whether any given False result is real or noise-induced.

Dually, an adversary that repeatedly ORs c with random values sees True for $c \in T$ (always True), but again noise inputs produce unpredictable outputs. For $c \in F$, the OR produces True only when $r_i \in T$ (5%), False when $r_i \in F$ (90%), and noise when $r_i \in N$ (5%). The noise fraction creates a floor of uncertainty in any statistical test the adversary performs.

Practical allocation. The cipher Boolean allocation $|T|/2^n = 0.05$, $|F|/2^n = 0.90$, $|N|/2^n = 0.05$ is motivated by the set membership use case. A cipher set $\hat{1}_A : \mathsf{C}(X) \to \mathsf{C}(\text{Bool})$ maps domain elements to cipher Booleans. Members are forced into T by the construction. Non-members land randomly: 90% decode as False (correct rejection), 5% as True (false positive), 5% as noise. The false positive rate equals p_{true} and is a system parameter, not dependent on the set size or universe.

Composition with set operations. Boolean search queries like "$w_1 \wedge w_2 \wedge \neg w_3$" are evaluated entirely on the untrusted machine by chaining cipher set evaluations through cipher AND/OR/NOT maps. Each Boolean operation is itself a cipher map (constructed via PHF over the $|T \cup F|^2$ input pairs for AND/OR, or $|T \cup F|$ for NOT). The untrusted machine sees opaque hash values flowing through opaque total functions at every stage.

7.3 Experimental Validation

A reference implementation (`cipher-maps`, Python with PHF backend via `phobic`) validates the construction on the 20 Newsgroups corpus (18,266 documents, 58,903 unique words). With $n = 8$ bits, $|T| = 13$, $|F| = 230$, the AND and OR cipher maps each have $(13 + 230)^2 = 59{,}049$ entries.

Boolean search at scale. Table 1 shows precision and recall for representative Boolean queries over 5,000 documents, with all Boolean operations evaluated on the untrusted machine. Construction rate: 843 documents per second.

Table 1: Boolean encrypted search (5,000 documents, $p_T = 0.05$).

Query	Precision	Recall	FP	Time
Single term	0.39	1.00	248	0.10s
2-term AND	0.26	1.00	42	0.30s
3-term AND	0.33	1.00	12	0.60s
OR	0.37	0.97	428	0.30s
OR AND NOT	0.33	0.88	389	0.55s

AND queries maintain perfect recall while false positives decrease (248 for single term, 12 for 3-term AND). OR and NOT queries lose recall (0.97 and 0.88) due to noise propagation: when a cipher set returns a noise value $n \in N$, the subsequent OR or NOT operation produces noise or an unpredictable result, effectively dropping the document from the result set.

FPR compounding vs. Bernoulli model. The Bernoulli composition model [17] predicts that k independent AND operations yield $\text{FPR}_k = p_T^k$. Empirically, OR chains match theory closely (ratio ≈ 0.9 across chain lengths 1–5), but AND chains diverge at depth > 2: the empirical FPR exceeds p_T^k by an order of magnitude at $k = 3$. The reason: each Bernoulli test is an independent draw, but cipher Boolean AND is a deterministic cipher map. After one AND, the result is a specific cipher Boolean value, and feeding it into the next AND produces a deterministic output. The independence assumption breaks at the cipher map level even though it holds within a single cipher set.

Encoding granularity. We built a 7-function loan approval pipeline at three granularity levels:

Level	Build time	Space	Intermediates
Root (1 cipher map)	0.009s	694 B	0
Intermediate (3 groups)	6.8s	—	3
Leaf (7 nodes)	5.9s	—	7

All three produce zero errors on the full domain (150 inputs). The root annotation is 650× faster (PHF backend vs. seed search) and exposes zero intermediate cipher values. The trade-off: the root annotation requires the entire input domain at construction time, while the leaf annotation builds smaller cipher maps that compose dynamically.

8 Discussion

Relationship to cipher maps. This paper extends the cipher map framework of [18] in two directions: inward (what happens when cipher values carry algebraic structure) and outward (what happens when the untrusted machine actively explores the cipher space). The cipher map paper focuses on individual maps and their four properties; this paper studies how those properties interact with type constructors and how confidentiality degrades under active use.

Relationship to Bernoulli data types. The accuracy-side analysis of algebraic types over approximate values is developed in [17], which studies the Kronecker factorization of joint confusion matrices for product types and the error propagation through sum-type constructors. That paper asks: how does η change when types are composed? This paper asks the complementary question: how does confidentiality change when types are composed? Together, they characterize the full trade-off space (accuracy × confidentiality) for algebraic types over trapdoor computing.

The encoding granularity principle. The product and sum trade-offs in Sections 4.2 and 4.3 are instances of a single principle developed in [18, Sec. 9]: coarser encoding granularity (encoding more values as a single unit) yields better confidentiality but fewer operations available to the untrusted machine. The entanglement parameter p [18, Sec. 9.3] controls the spectrum: $p = 1$ (component-wise) gives maximum functionality and minimum confidentiality; $p = k$ (whole-state) gives maximum confidentiality and minimum functionality.

Open questions.

1. *Recursive types.* Lists, trees, and other recursive types are built from sums and products. The sum-type impossibility (Theorem 4.2) implies that recursive traversal (which requires pattern matching at each node) either leaks the structure (which constructors are used at each level) or requires a trusted-machine round trip at each node. Quantifying the total leakage for a recursive traversal of depth d is open.

2. *Tight orbit bounds for specific constructions.* Theorem 5.3 gives a general upper bound on confidentiality in terms of orbit size. For specific cipher map constructions (e.g., the trapdoor boolean algebra of [18]), tighter bounds may be possible by exploiting the structure of the construction.

3. *Cipher TM with oblivious head.* Can the head direction be ciphered at acceptable cost? The naive approach (evaluate all three possible next states) triples the work per step. Whether

a sub-multiplicative overhead exists, achieving partial head-movement hiding without full ORAM cost, is open.

4. *Partial sum-type hiding.* Theorem 4.2 shows that full tag hiding and full untrusted pattern matching are mutually exclusive. Is there a meaningful intermediate notion—hiding the tag from some adversary classes while permitting restricted pattern matching? For instance, one could imagine a construction where the tag is hidden against passive adversaries (who only observe cipher values) but available to active adversaries (who evaluate cipher maps).

5. *Cut-point placement as an optimization.* Section 6.3 describes cut points as a design knob spanning cipher-TM-style iteration and leaf-level expression-tree annotation. Formulating cut-point selection as a constrained optimization, subject to the orbit bound of Proposition 5.5 and construction-time space budgets, would turn the choice into a solvable problem rather than a designer's judgement call.

9 Conclusion

We have shown that algebraic type constructors create unavoidable confidentiality costs in trapdoor computing. The sum-type impossibility theorem establishes that tag hiding and untrusted pattern matching are mutually exclusive for any encoding. The orbit closure framework provides a quantitative bound on confidentiality degradation as the untrusted machine applies operations, and a typed-composition discipline converts that bound into a design-time budget. These results apply to any cipher map construction, not just specific implementations.

Two realizations concretize these trade-offs. Expression-tree decomposition with cipher-node annotations lets the programmer choose the granularity at which intermediate cipher values are exposed; a cipher Turing machine gives space $O(|Q| \cdot |\Gamma|)$ at the cost of leaking head-movement patterns. Both are instances of a single cut-point pattern that the typed-chain bound constrains identically.

On the practical side, cipher Boolean types with AND/OR/NOT as cipher maps enable Boolean search queries evaluated entirely on the untrusted machine. Experiments on the 20 Newsgroups corpus (5,000 documents) show perfect recall for AND queries, with false positive rates matching theoretical predictions for single-term queries and compounding predictably through Boolean chains. The encoding granularity spectrum is measurable: root annotation (one cipher map for the entire program) is 650× faster and exposes zero intermediates compared to leaf annotation (one cipher map per function). The gap between the Bernoulli independence model and the cipher map implementation for AND chain composition points to future work on tighter error models for composed cipher maps.

Acknowledgments

The algebraic cipher types concept originates in a 2019–2022 C++ notebook by the author, which explored cipher Booleans, cipher unions, and cipher Turing machines through concrete implementations. The present paper extracts and formalizes the theoretical content from that notebook, building on the cipher map framework developed in [18].

References

[1] 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.

[2] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273, 2011.

[4] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, pages 668–679, 2015.

[5] 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.

[6] 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.

[7] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[8] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[9] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why your encrypted database is not secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 162–168, 2017.

[10] 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.

[11] Ken Lang. NewsWeeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339, 1995.

[12] 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.

[13] Andrei Sabelfeld and Andrew C Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[14] Claude E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[15] Geoffrey Smith. On the foundations of quantitative information flow. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, pages 288–302, 2009.

[16] 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.

[17] Alexander Towell. Bernoulli data types: Approximate algebraic types with quantitative error, 2026. Manuscript in preparation.

[18] Alexander Towell. Cipher maps: Total functions as trapdoor approximations, 2026. Manuscript in preparation.

[19] Alexander Towell. Maximizing confidentiality under trapdoor computing: Encoding granularity and optimal space, 2026. Manuscript in preparation.

[20] Alan M Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1936.

[21] Andrew C Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.