Oblivious Computing: Privacy Through Approximation

From Uniform Distributions to Information-Theoretic Security

Alexander Towell
atowell@siue.edu

January 24, 2026

Abstract

Computing on hidden data while limiting observable information leakage is a fundamental challenge in secure computation. We present *oblivious types*, a framework where privacy emerges from uniform observable distributions. The central thesis is simple: *uniformity equals privacy*—when all observations are equally likely regardless of the hidden value, no information is leaked.

This paper develops the theory of oblivious types systematically. We establish a three-level hierarchy of obliviousness (perfect, statistical, computational) and reveal a fundamental tension: functional consistency necessarily creates correlations that break perfect uniformity—the *random oracle paradox*. However, we show that approximate computation provides a principled resolution: Bernoulli types, with their controlled error rates, naturally provide "plausible deniability" that bounds information leakage.

The key technical contribution is the *uniform encoding construction*: by sizing encoding sets inversely proportional to output frequency, we achieve maximum entropy without explicit randomization. Invalid inputs become a "river of noise" that provides privacy for free. We formalize this through rank-deficient confusion matrices, proving that privacy emerges from information loss rather than added randomness.

The framework unifies diverse privacy-preserving constructions—from oblivious RAM to secure indexes—under one theoretical lens, providing both understanding of fundamental limits and practical guidance for system design.

1 Introduction

1.1 The Access Pattern Problem

Computing on sensitive data requires more than encrypting data at rest. Even with perfect encryption, the *pattern* of data access can leak information. A database encrypted with state-of-the-art cryptography still reveals which records are accessed together, how often each location is touched, and whether the same query is repeated. This *access pattern leakage* has enabled devastating attacks on encrypted databases, cloud storage, and outsourced computation.

Consider a medical records database. Even if record contents are encrypted, an adversary observing access patterns might learn:

- Which patients have similar conditions (same records accessed together)
- Disease prevalence (frequency of accessing certain records)
- Query correlations (searches that always coincide)

The fundamental problem is that *computation is observable*. Any deterministic function must produce the same output for the same input, creating correlations that leak information about inputs. This paper develops a framework for understanding and managing this leakage.

1.2 Uniformity as Privacy

Our central thesis is deceptively simple:

Principle 1.1 (Uniformity = Privacy)**.** If the observable trace of a computation is uniformly distributed over all possible traces, independent of the hidden input, then the computation leaks no information.

This principle transforms privacy from a cryptographic property into an information-theoretic one. Rather than asking "can an adversary break the encryption?" we ask "does the observable distribution depend on the hidden value?"

Formally, for a hidden value h and observable trace π:

$$\text{Perfect Privacy} \iff \forall h_1, h_2 : \quad \mathbb{P}[\pi \mid h_1] = \mathbb{P}[\pi \mid h_2] \tag{1}$$

This view unifies diverse privacy mechanisms. Oblivious RAM achieves privacy by making access patterns independent of the access sequence. Differential privacy adds noise to make outputs statistically close regardless of individual records. Our framework captures these as instances of the uniformity principle.

1.3 The Random Oracle Paradox

A crucial challenge immediately arises. Consider a deterministic function f:

- **Functional requirement**: $f(x) = f(x)$ always (same input, same output)
- **Privacy requirement**: $\mathbb{P}[\pi \mid x_1] = \mathbb{P}[\pi \mid x_2]$ (traces independent of input)

These requirements conflict fundamentally. If two different inputs $x_1 \neq x_2$ have $f(x_1) = f(x_2)$, their traces must correlate. But if $f(x_1) \neq f(x_2)$, their traces must differ. Either way, *functional consistency creates correlations that break perfect uniformity.*

We call this the *random oracle paradox*: a truly random function would provide perfect privacy (each output independent), but real functions must maintain input-output relationships. This tension is fundamental—not a limitation of specific constructions but an inherent property of deterministic computation.

1.4 Approximation as Resolution

The resolution comes from an unexpected direction: *approximation*. By allowing computations to produce incorrect results with controlled probability, we create "plausible deniability" that bounds information leakage.

Consider a Bloom filter testing membership in a set S:

- For $x \in S$: always returns true (no false negatives)
- For $x \notin S$: returns true with probability α (false positives)

The false positives are not a bug—they are a privacy feature. When the filter returns true, an adversary cannot distinguish between "x is definitely in S" and "x is not in S, but the filter erred." This ambiguity limits information leakage.

This paper develops this insight systematically. Bernoulli types (developed in our companion paper [**?**]) provide controlled approximation. Oblivious types, developed here, leverage that approximation for privacy. The combination provides a powerful framework for privacy-preserving computation.

1.5 Contributions

This paper makes the following contributions:

1. **Oblivious Types** (§2): We formalize the hidden/observable duality and establish a three-level hierarchy of obliviousness (perfect, statistical, computational).
2. **The Random Oracle Paradox** (§3): We prove that functional consistency fundamentally limits perfect obliviousness, explaining why practical systems must accept bounded leakage.
3. **Information-Theoretic Framework** (§4): We develop rank-deficient confusion matrices as the mathematical foundation, proving that privacy emerges from information loss.
4. **Leakage Decomposition** (§5): We show how Bernoulli types compose with oblivious types, separating access pattern leakage from result approximation leakage.
5. **Uniform Encoding Construction** (§6): We present the key technical contribution: sizing encoding sets inversely to frequency achieves optimal uniformity.
6. **Applications** (§7): We apply the framework to secure indexes and oblivious data structures.

1.6 Paper Organization

Section 2 formalizes oblivious types and the obliviousness hierarchy. Section 3 analyzes the fundamental tension between functionality and privacy. Section 4 develops the information-theoretic framework. Section 5 shows how approximation provides privacy. Section 6 presents the uniform encoding construction. Section 7 applies the theory to practical constructions. Section 8 discusses related work, and Section 9 concludes.

2 Oblivious Types

We now formalize the hidden/observable duality that underlies privacy-preserving computation.

2.1 The Hidden/Observable Duality

In Bernoulli types, we distinguished *latent* values (true mathematical objects) from *observed* values (what computations produce). Oblivious types introduce a parallel distinction:

- **Hidden values**: The secret data we wish to protect—query contents, database records, computation inputs.
- **Observable traces**: What an adversary can see—access patterns, timing, encrypted ciphertext lengths.

The key difference is perspective. In Bernoulli types, the "observer" is the program itself, receiving approximate results. In oblivious types, the "observer" is an adversary, seeing the side-effects of computation.

Definition 2.1 (Oblivious Type)**.** An *oblivious type* $\mathcal{O}\langle T \rangle$ consists of:

1. A *hidden value space* T containing the secrets we protect
2. A *trace space* Π of observable behaviors
3. A *leakage function* $\mathcal{L}(\cdot) : T \to \Pi$ mapping hidden values to observable traces

The leakage function captures what an adversary learns by observing the computation. Perfect privacy means the leakage function is constant—all inputs produce indistinguishable traces.

Example 2.2 (Encrypted Database Query)**.** For a query on an encrypted database:

- Hidden value: The query predicate ("find all patients with condition X")
- Observable trace: The set of encrypted record IDs returned
- Leakage function: Maps query to the access pattern

Even without decrypting records, the adversary learns which records match the query.

2.2 The Obliviousness Hierarchy

Not all oblivious types provide the same privacy guarantees. We distinguish three levels:

Definition 2.3 (Perfect Obliviousness)**.** An oblivious type $\mathcal{O}\langle T \rangle$ is *perfectly oblivious* if the trace is independent of the hidden value:

$$\forall h_1, h_2 \in T : \quad \mathbb{P}\left[\pi = \pi \mid h = h_1\right] = \mathbb{P}\left[\pi = \pi \mid h = h_2\right] \tag{2}$$

Equivalently, $I(h; \pi) = 0$.

Definition 2.4 (Statistical Obliviousness)**.** An oblivious type is *ϵ-statistically oblivious* if trace distributions are close:

$$\forall h_1, h_2 \in T : \quad \sum_{\pi \in \Pi} \left|\mathbb{P}\left[\pi = \pi \mid h_1\right] - \mathbb{P}\left[\pi = \pi \mid h_2\right]\right| \leq \epsilon \tag{3}$$

For security parameter λ, we require $\epsilon = \mathsf{negl}(\lambda)$.

Definition 2.5 (Computational Obliviousness)**.** An oblivious type is *computationally oblivious* if no PPT adversary can distinguish traces:

$$\forall \mathsf{PPT}\ \mathcal{A} : \quad \left|\mathbb{P}\left[\mathcal{A}(\pi(h_1)) = 1\right] - \mathbb{P}\left[\mathcal{A}(\pi(h_2)) = 1\right]\right| \leq \mathsf{negl}(\lambda) \tag{4}$$

The hierarchy is strict: perfect $\Rightarrow$ statistical $\Rightarrow$ computational. Practical systems typically achieve computational obliviousness, with statistical or perfect being too expensive or impossible.

Example 2.6 (Oblivious RAM)**.** Oblivious RAM (ORAM) achieves computational obliviousness for memory access patterns. Given a sequence of read/write operations, ORAM produces access traces computationally indistinguishable from random. However, it incurs $O(\log N)$ overhead per operation—the price of hiding access patterns.

2.3 Privacy as Mutual Information

The information-theoretic view quantifies privacy precisely:

Definition 2.7 (Privacy Loss)**.** For hidden value h and trace π, the *privacy loss* is:

$$\text{Privacy Loss} = I(h; \pi) = H(h) - H(h \mid \pi) \tag{5}$$

Mutual information measures how much observing the trace reduces uncertainty about the hidden value. Perfect obliviousness means $I(h; \pi) = 0$—the trace tells the adversary nothing. Maximum leakage means $I(h; \pi) = H(h)$—the trace completely determines the hidden value.

Proposition 2.8 (Bounds on Privacy Loss)**.** *For any oblivious type:*

$$0 \leq I(h; \pi) \leq \min(H(h), H(\pi)) \tag{6}$$

Proof. The lower bound is the definition of mutual information being non-negative. The upper bound follows from mutual information being bounded by either marginal entropy. ■

2.4 Composition of Oblivious Types

Operations on oblivious types must preserve privacy. We analyze how composition affects leakage.

Theorem 2.9 (Leakage Under Composition)**.** *For oblivious types $\mathcal{O}\langle A\rangle$ and $\mathcal{O}\langle B\rangle$ with independent hidden values, their composition has leakage bounded by:*

$$I((a, b); \pi_{A \circ B}) \leq I(a; \pi_A) + I(b; \pi_B) \tag{7}$$

Proof. By the chain rule for mutual information and independence of hidden values. The inequality becomes equality when traces are independent given hidden values. ■

This theorem justifies modular design: if each component has bounded leakage, their composition has bounded (additive) leakage. However, as we will see in Section 3, *dependent* compositions can leak more due to correlations.

3 The Random Oracle Paradox

We now analyze the fundamental tension between functionality and privacy.

3.1 Consistency versus Uniformity

A deterministic function $f : X \to Y$ has an inherent tension:

- **Consistency**: Same input must produce same output: $f(x) = f(x)$
- **Uniformity**: Output distribution should be independent of input

These requirements conflict. If f is consistent, then querying the same input twice produces identical outputs—a correlation. An adversary observing "same output twice" learns "same input twice."

Theorem 3.1 (Correlation Leakage)**.** *Any deterministic function $f : X \to Y$ leaks equality information:*

$$f(x_1) = f(x_2) \Rightarrow \textit{Traces correlated} \tag{8}$$

This leakage is inherent and cannot be eliminated without either:

1. *Abandoning consistency (randomized outputs)*
2. *Accepting information loss (many-to-one mapping)*

Proof. If $f(x_1) = f(x_2)$, an observer seeing output y twice knows both queries produced the same result. If the adversary knows f, they learn $f^{-1}(y)$ is queried twice. For injective f, this reveals $x_1 = x_2$.

Randomization breaks consistency. Many-to-one mappings introduce ambiguity but also information loss. ■

3.2 The Paradox Stated

Principle 3.2 (Random Oracle Paradox)**.** A truly random oracle $\mathcal{R} : X \to Y$ (fresh random output for each new query, consistent for repeated queries) would provide near-perfect obliviousness for single queries but increasingly leaks under composition:

1. Single query: Output uniformly random, independent of input
2. Two queries: Output pair reveals whether inputs equal
3. n queries: Output sequence reveals full equality pattern

No deterministic construction can achieve the single-query property while avoiding the correlation leakage.

This paradox explains why practical oblivious systems must make trade-offs. ORAM uses $O(\log N)$ overhead to rerandomize accesses. Oblivious algorithms pad and shuffle to hide patterns. All approaches pay a cost for hiding correlations.

3.3 Correlation Propagation

Correlations compound through composition. Consider a sequence of operations:

Example 3.3 (Query Sequence Correlation)**.** An adversary observes encrypted search queries $q_1, q_2, \ldots, q_n$:

1. If $q_i = q_j$, results are identical (perfect correlation)
2. If queries share keywords, results overlap (partial correlation)
3. Query frequency reveals popularity distribution

Over time, correlations accumulate until the adversary can deduce the query distribution.

Theorem 3.4 (Correlation Accumulation)**.** *For n operations with trace $\pi_1, \ldots, \pi_n$, the total leakage satisfies:*

$$I((h_1, \ldots, h_n); (\pi_1, \ldots, \pi_n)) \geq I((h_1, \ldots, h_n); \textit{Equality Pattern}) \tag{9}$$

where the equality pattern records which $h_i = h_j$.

Proof. The equality pattern is determined by the traces (same trace implies same hidden value for deterministic operations). Mutual information with any function of traces is at most mutual information with traces. Since equality is derivable from traces, the inequality follows. ■

3.4 Whole-Program versus Compositional Obliviousness

The correlation problem leads to two fundamentally different approaches:

Definition 3.5 (Whole-Program Obliviousness)**.** A computation is *whole-program oblivious* if the entire trace sequence is oblivious with respect to all inputs simultaneously. This requires hiding inter-operation correlations.

Definition 3.6 (Compositional Obliviousness)**.** A computation is *compositionally oblivious* if each operation's trace is individually oblivious. This does not hide correlations between operations.

Whole-program obliviousness is stronger but more expensive:

Proposition 3.7 (Whole-Program Cost)**.** *Achieving whole-program obliviousness for n operations on N items requires $\Omega(n \log N)$ work (the ORAM lower bound).*

Compositional obliviousness is cheaper but leaks correlations:

Proposition 3.8 (Compositional Leakage)**.** *Compositionally oblivious operations leak the equality pattern: which operations had the same inputs.*

This trade-off is fundamental. The next sections show how approximation helps navigate it.

4 Information-Theoretic Framework

We now develop the mathematical foundations connecting confusion matrices to privacy.

4.1 Confusion Matrices for Privacy

Recall from Bernoulli types that a confusion matrix Q relates latent to observed values. For oblivious types, it relates hidden values to observable traces:

Definition 4.1 (Privacy Confusion Matrix)**.** For hidden values in $\{h_1, \ldots, h_n\}$ and traces in $\{\pi_1, \ldots, \pi_m\}$, the *privacy confusion matrix* $Q \in \mathbb{R}^{n \times m}$ has entries:

$$q_{ij} = \mathbb{P}\left[\pi = \pi_j \mid h = h_i\right] \tag{10}$$

Each row is a probability distribution: $\sum_j q_{ij} = 1$.

The confusion matrix encodes everything about leakage. Different matrix structures correspond to different privacy properties:

Proposition 4.2 (Matrix Structure and Privacy)**.**

1. *Perfect obliviousness $\iff$ all rows identical*
2. *No privacy (full leakage) $\iff$ Q is a permutation matrix*
3. *Partial privacy $\iff$ Q has rank between 1 and* $\min(n, m)$

Proof. (1) All rows identical means $\mathbb{P}\left[\pi \mid h_1\right] = \mathbb{P}\left[\pi \mid h_2\right]$ for all h_1, h_2. (2) Permutation matrix means each hidden value produces a unique trace. (3) Intermediate rank means some but not all distinctions are preserved. ■

4.2 Rank Deficiency as Privacy

The key insight is that *rank deficiency creates privacy*:

Theorem 4.3 (Rank Deficiency Privacy)**.** *For confusion matrix Q with rank $r < n$:*

1. *The kernel* $\ker(Q^\top)$ *has dimension* $n - r$
2. *Hidden values in the same coset of* $\ker(Q^\top)$ *are indistinguishable*
3. *Privacy emerges from the equivalence classes, not added randomness*

Proof. If Q has rank $r < n$, its row space has dimension r. The left null space (row-equivalent vectors) has dimension $n - r$. Two hidden values h_i, h_j are indistinguishable if rows i and j of Q are identical, placing them in the same equivalence class. ■

This theorem provides a different perspective on privacy: rather than adding noise to obscure information, we *lose information* through rank-deficient transformations. The "noise" is not added—it emerges from the structure of the observation itself.

Corollary 4.4 (Privacy Parameter)**.** *For confusion matrix Q with rank r, define the privacy parameter:*

$$\epsilon = \ln\left(\frac{n}{r}\right) \tag{11}$$

where n is the number of hidden values. Higher ϵ means more privacy (more values per equivalence class).

4.3 Entropy and Obliviousness

Entropy provides a scalar measure of obliviousness:

Definition 4.5 (Observable Entropy)**.** The *observable entropy* of an oblivious type is:

$$H(\pi) = -\sum_{\pi \in \Pi} \mathbb{P}\left[\pi = \pi\right] \log \mathbb{P}\left[\pi = \pi\right] \tag{12}$$

Definition 4.6 (Maximum Observable Entropy)**.** The *maximum observable entropy* is $H_{\max} = \log |\Pi|$, achieved when all traces are equally likely.

Definition 4.7 (Obliviousness Ratio)**.** The *obliviousness ratio* measures how close to maximum entropy:

$$\rho = \frac{H(\pi)}{H_{\max}} = \frac{H(\pi)}{\log |\Pi|} \tag{13}$$

where $\rho = 1$ is perfect uniformity and $\rho < 1$ indicates deviation.

Theorem 4.8 (Compression Test for Obliviousness)**.** *If traces can be compressed below their maximum entropy:*

$$\textit{Compressed size} < H_{\max} \tag{14}$$

then the system is not perfectly oblivious. The compression ratio bounds the privacy loss.

Proof. By Shannon's source coding theorem, data with entropy H cannot be compressed below H bits on average. If traces compress to $H' < H_{\max}$, then $H(\pi) \leq H' < H_{\max}$, implying non-uniform distribution. ■

This theorem enables *empirical privacy testing*: compress observed traces and check if they compress significantly below the maximum. Significant compression reveals pattern leakage.

4.4 Mutual Information Decomposition

We can decompose privacy loss into interpretable components:

Theorem 4.9 (Mutual Information Chain Rule)**.** *For hidden value h and trace π:*

$$I(h; \pi) = H(\pi) - H(\pi \mid h) \tag{15}$$

Privacy loss equals trace entropy minus conditional entropy given the hidden value.

For deterministic functions where $\pi = f(h)$:

$$H(\pi \mid h) = 0 \implies I(h; \pi) = H(\pi) \tag{16}$$

This confirms that for deterministic functions, observable entropy equals leakage. To reduce leakage, we must either:

1. Reduce $H(\pi)$ (fewer possible traces, coarser observation)
2. Increase $H(\pi \mid h)$ (add randomness)

Bernoulli types achieve the first: by producing approximate results, we collapse many inputs into fewer observable outcomes.

5 Approximation as Privacy Mechanism

We now show how Bernoulli types provide privacy through controlled approximation.

5.1 The Composition of Bernoulli and Oblivious Types

Bernoulli types approximate values; oblivious types hide them. Combined, they provide both:

Definition 5.1 (Oblivious Bernoulli Type)**.** An *oblivious Bernoulli type* $\mathcal{O}\langle\mathcal{B}\langle T\rangle\rangle$ is:

- Hidden: A latent value $t \in T$
- Observed: An approximate value $\tilde{t}$ that may differ from t
- Observable: The trace π of accessing/computing $\tilde{t}$

Both approximation error and access pattern leakage are controlled.

Example 5.2 (Oblivious Bloom Filter)**.** An oblivious Bloom filter is $\mathcal{O}\langle\mathcal{B}\langle\text{Set}\rangle\rangle$:

- Hidden: The true set S
- Observed: Membership query results (may false positive)
- Observable: Which bit positions are accessed

Even if access patterns leak some information, false positives create uncertainty about the true set.

5.2 Leakage Decomposition

The key theorem separates two sources of leakage:

Theorem 5.3 (Leakage Decomposition)**.** *For oblivious Bernoulli type $\mathcal{O}\langle\mathcal{B}\langle T\rangle\rangle$ with hidden value t, approximate result $\tilde{t}$, and trace π:*

$$\underbrace{I(t;\pi,\tilde{t})}_{\textit{Total Leakage}} = \underbrace{I(t;\pi)}_{\textit{Access Pattern Leakage}} + \underbrace{I(t;\tilde{t}\mid\pi)}_{\textit{Result Leakage}} \tag{17}$$

Proof. By the chain rule for mutual information:

$$I(t;\pi,\tilde{t}) = I(t;\pi) + I(t;\tilde{t}\mid\pi) \tag{18}$$

The first term is leakage from the access pattern alone. The second is additional leakage from the approximate result given the access pattern. ■

This decomposition is powerful because the two leakage sources can be optimized independently:

- **Access pattern leakage**: Controlled by oblivious construction (padding, shuffling, ORAM)
- **Result leakage**: Controlled by Bernoulli error rates (false positives/negatives)

5.3 False Positives as Plausible Deniability

False positives create ambiguity that limits information leakage:

Theorem 5.4 (Plausible Deniability from False Positives)**.** *For a Bernoulli set $\tilde{S}$ with false positive rate α and latent set S, observing $\tilde{S}.\mathsf{contains}(x) = \mathsf{true}$ gives:*

$$\mathbb{P}\left[x \in S \mid \mathsf{true}\right] = \frac{\mathbb{P}\left[x \in S\right]}{\mathbb{P}\left[x \in S\right] + \alpha \cdot \mathbb{P}\left[x \notin S\right]} \tag{19}$$

When $\alpha > 0$, the adversary cannot be certain $x \in S$.

Proof. By Bayes' theorem:

$$\mathbb{P}\left[x \in S \mid \mathsf{true}\right] = \frac{\mathbb{P}\left[\mathsf{true} \mid x \in S\right] \cdot \mathbb{P}\left[x \in S\right]}{\mathbb{P}\left[\mathsf{true}\right]} \tag{20}$$

$$= \frac{1 \cdot \mathbb{P}\left[x \in S\right]}{\mathbb{P}\left[x \in S\right] + \alpha \cdot \mathbb{P}\left[x \notin S\right]} \tag{21}$$

where we use $\mathbb{P}\left[\mathsf{true} \mid x \in S\right] = 1$ (no false negatives for standard Bloom filters). ■

Corollary 5.5 (Privacy from Error Rates)**.** *Higher false positive rates provide more privacy:*

$$\lim_{\alpha \to 1} \mathbb{P}\left[x \in S \mid \mathsf{true}\right] = \mathbb{P}\left[x \in S\right] \tag{22}$$

At $\alpha = 1$, the result carries no information (always true regardless of membership).

This corollary reveals a fundamental trade-off: more error means more privacy but less utility. The Bernoulli framework makes this trade-off explicit and controllable.

5.4 Bounding Result Leakage

The second term in the leakage decomposition can be bounded by error rates:

Theorem 5.6 (Result Leakage Bound)**.** *For Bernoulli boolean $\tilde{b}$ with rates (α, β):*

$$I(b; \tilde{b}) \leq 1 - h_2\left(\frac{\alpha + \beta}{2}\right) \tag{23}$$

where $h_2(p) = -p \log p - (1-p)\log(1-p)$ is binary entropy.

Proof. The mutual information between b and $\tilde{b}$ is bounded by their correlation. For a binary symmetric channel with crossover probability $p = (\alpha + \beta)/2$, the capacity is $1 - h_2(p)$. ■

Corollary 5.7 (Zero Leakage Condition)**.** *Result leakage is zero when $\alpha + \beta = 1$:*

$$\alpha + \beta = 1 \implies I(b; \tilde{b}) = 0 \tag{24}$$

This occurs when the observation is independent of the truth (random guess).

The corollary confirms intuition: when the error is so high that the result is effectively random, no information is leaked. This extreme case sacrifices all utility for perfect privacy.

5.5 Optimizing the Trade-off

Practical systems operate between extremes. Given utility requirements (maximum acceptable error), we can determine the privacy budget:

Proposition 5.8 (Privacy-Utility Trade-off)**.** *For a Bernoulli type with error rates (α, β):*

1. *Utility requirement: $\alpha + \beta \leq \tau$ (error tolerance)*
2. *Privacy budget: $I(b; \tilde{b}) \leq 1 - h_2(\tau/2)$*

Lower error tolerance means higher privacy budget required.

This makes the privacy-utility trade-off quantitative: we can compute exactly how much information leakage a given accuracy requirement allows.

6 Uniform Encoding Construction

We now present the key technical contribution: achieving privacy through encoding design rather than explicit randomization.

6.1 The Inverse-Frequency Principle

Recall from Bernoulli types the hash-based construction: values are represented by seeds mapping to valid encodings. The key insight for privacy is:

Principle 6.1 (Inverse-Frequency Encoding)**.** For output y with frequency $\mathsf{Freq}(y)$, set the encoding set size:

$$|\mathsf{Valid}(y)| = \frac{c}{\mathsf{Freq}(y)} \tag{25}$$

for constant c. This achieves uniform output distribution.

Theorem 6.2 (Optimal Uniformity)**.** *With inverse-frequency encoding, the output distribution is uniform:*

$$\mathbb{P}\left[\mathit{Output} = y\right] = \mathbb{P}\left[h(\mathit{input}) \in \mathsf{Valid}(y)\right] = \frac{|\mathsf{Valid}(y)|}{|\mathit{Hash\ Range}|} \cdot \mathsf{Freq}(y) = \frac{c}{|\mathit{Hash\ Range}|} \tag{26}$$

which is constant for all y.

Proof. For a random hash:

$$\mathbb{P}\left[\text{Output} = y\right] = \sum_x \mathbb{P}\left[\text{Input} = x\right] \cdot \mathbb{P}\left[h(x) \in \mathsf{Valid}(y)\right] \tag{27}$$

With $\mathbb{P}\left[h(x) \in \mathsf{Valid}(y)\right] \propto |\mathsf{Valid}(y)|$ and $|\mathsf{Valid}(y)| \propto 1/\mathsf{Freq}(y)$:

$$\mathbb{P}\left[\text{Output} = y\right] \propto \mathsf{Freq}(y) \cdot \frac{1}{\mathsf{Freq}(y)} = 1 \tag{28}$$

■

This theorem is remarkable: by carefully sizing encoding sets, we achieve uniform outputs *without adding explicit randomness*. The hash function's natural behavior, combined with proper encoding, yields uniformity automatically.

6.2 Invalid Inputs as Free Noise

The encoding construction has a beautiful property for privacy:

Definition 6.3 (Invalid Input)**.** An input x is *invalid* for output y if $h(x) \notin \mathsf{Valid}(y)$ for all valid y. That is, the hash falls outside all valid encoding sets.

Theorem 6.4 (Free Noise from Invalid Inputs)**.** *Invalid inputs naturally produce uniform outputs without affecting valid computations:*

1. *For valid input (x in domain): Output is determined by encoding lookup*
2. *For invalid input (x not in domain): Output is uniformly random (falls in no valid set)*

Invalid inputs provide "noise" that masks valid outputs, for free.

Proof. For invalid x, $h(x)$ is uniform over the hash range. The probability of falling in $\mathsf{Valid}(y)$ is proportional to $|\mathsf{Valid}(y)|$. With inverse-frequency sizing, this gives uniform distribution over outputs.

For valid x, the seed was chosen to ensure $h(x) \in \mathsf{Valid}(f(x))$, so the correct output is returned. ■

This theorem explains a key advantage of hash-based constructions: the "river of noise" from invalid inputs naturally provides obliviousness. We don't need to generate fake queries or pad with dummies—the construction provides noise automatically.

6.3 Contrast with Space-Optimal Encoding

In Bernoulli types, we sized encoding sets *proportionally* to frequency for space efficiency:

$$|\mathsf{Valid}(y)| \propto \mathsf{Freq}(y) \quad \text{(Bernoulli: space-optimal)} \tag{29}$$

For oblivious types, we use *inverse-proportional* sizing for uniformity:

$$|\mathsf{Valid}(y)| \propto \frac{1}{\mathsf{Freq}(y)} \quad \text{(Oblivious: privacy-optimal)} \tag{30}$$

Theorem 6.5 (Encoding Strategy Trade-off)**.** *The same hash-based construction achieves:*

1. *Space-optimal:* $|\mathsf{Valid}(y)| \propto \mathsf{Freq}(y)$ *minimizes expected encoding size*
2. *Privacy-optimal:* $|\mathsf{Valid}(y)| \propto 1/\mathsf{Freq}(y)$ *maximizes output entropy*

These are endpoints of a spectrum; intermediate strategies trade off space and privacy.

Proof. For space: Expected encoding size is $\mathbb{E}\left[|\mathsf{Valid}(Y)|\right] = \sum_y \mathsf{Freq}(y) \cdot |\mathsf{Valid}(y)|$. Minimized when $|\mathsf{Valid}(y)| \propto \mathsf{Freq}(y)$ (by convexity argument).

For privacy: Output entropy is maximized at $\log |\mathcal{Y}|$ when all outputs equally likely. Achieved when $\mathbb{P}\left[Y = y\right]$ is constant, which requires $|\mathsf{Valid}(y)| \propto 1/\mathsf{Freq}(y)$. ■

This theorem reveals the deep connection between Bernoulli and oblivious types: they use the same construction with different encoding strategies. Bernoulli types optimize for space; oblivious types optimize for privacy. Both are instances of the universal hash-based framework.

6.4 Practical Implementation

The uniform encoding construction translates to a practical algorithm:

Input: Frequency distribution $\mathsf{Freq}(\cdot)$, output space $\mathcal{Y}$, hash range $[0, R)$
Output: Encoding intervals for each output
$\mathsf{offset} \leftarrow 0$;
foreach $y \in \mathcal{Y}$ *sorted by frequency* **do**
  $\mathsf{size}_y \leftarrow \lfloor R/(|\mathcal{Y}| \cdot \mathsf{Freq}(y)) \rfloor$;
  $\mathsf{Valid}(y) \leftarrow [\mathsf{offset}, \mathsf{offset} + \mathsf{size}_y)$;
  $\mathsf{offset} \leftarrow \mathsf{offset} + \mathsf{size}_y$;
end
return $\{\mathsf{Valid}(y)\}_{y \in \mathcal{Y}}$;

Algorithm 1: Uniform Encoding Construction

Proposition 6.6 (Construction Efficiency)**.** *Algorithm 1 runs in $O(|\mathcal{Y}| \log |\mathcal{Y}|)$ time (dominated by sorting) and produces encodings that achieve obliviousness ratio $\rho \geq 1 - O(1/R)$.*

6.5 Composition Preserves Uniformity

A critical property: uniform encoding composes well.

Theorem 6.7 (Composition Uniformity)**.** *For uniformly-encoded oblivious types $\mathcal{O}\langle A \rangle$ and $\mathcal{O}\langle B \rangle$ composed as $\mathcal{O}\langle A \times B \rangle$:*

$$\textit{If } \mathbb{P}\left[A = a\right] = \mathbb{P}\left[B = b\right] = \frac{1}{|\mathcal{A}|} = \frac{1}{|\mathcal{B}|} \tag{31}$$

then

$$\mathbb{P}\left[(A, B) = (a, b)\right] = \frac{1}{|\mathcal{A}| \cdot |\mathcal{B}|} \tag{32}$$

Product uniformity is preserved.

Proof. For independent uniformly-encoded components, the product distribution is the product of marginals. Each marginal is uniform by construction. ■

This theorem enables modular design: build uniform components, compose them freely, maintain uniformity throughout.

6.6 Refresh Timing: The MAB Method

A critical operational question: *when should oblivious encodings be refreshed?* Even perfectly uniform encodings eventually leak information as adversaries accumulate observations. The Moving Average Bootstrap (MAB) method [**?**] provides a principled answer.

Definition 6.8 (Adversary Success Threshold)**.** Let N^* be the minimum number of encrypted operations an adversary must observe to achieve decoding accuracy p^* with confidence θ. Formally, N^* is the smallest n such that:

$$\mathbb{P}\left[\text{Adversary accuracy} \geq p^* \mid n \text{ observations}\right] \geq \theta \tag{33}$$

The defender's strategy is to refresh encodings *before* the adversary reaches N^* observations.

Theorem 6.9 (MAB Estimation Efficiency)**.** *The Moving Average Bootstrap method estimates N^* using only observed query patterns, achieving:*

1. *Estimation time: approximately 5% of direct calculation*
2. *No need to wait for N^* legitimate queries*
3. *Adaptive refinement as more data arrives*

Construction 6.10 (Adaptive Re-randomization Schedule)**.** Using the MAB method for proactive defense:

1. Monitor query/operation stream continuously
2. Estimate adversary's accumulated information using bootstrap sampling
3. When estimated adversary confidence exceeds threshold τ:
   - Generate new encoding functions (fresh hash seeds)
   - Refresh all valid encoding sets
   - Re-encode stored data if necessary
4. Adjust monitoring window based on observed attack patterns

Remark 6.11 (Proactive vs. Reactive Defense)*.* The MAB method enables *proactive* defense: refresh before the adversary succeeds, not after an attack is detected. This is particularly important because:

- Successful attacks may be silent (adversary learns without triggering alerts)

- Reactive detection requires observable attack signatures
- By the time an attack is detected, information has already leaked

The cost of proactive refresh (re-encoding overhead) is typically much lower than the cost of information leakage.

Proposition 6.12 (Refresh-Privacy Trade-off)**.** *More frequent encoding refresh provides better privacy but incurs overhead:*

$$\textit{Privacy} \propto \frac{1}{\textit{Average observations before refresh}} \tag{34}$$

The optimal refresh frequency balances privacy requirements against re-encoding costs.

This timing mechanism completes the operational picture: uniform encoding provides privacy at each moment, and the MAB method determines when to restore uniformity as observations accumulate.

7 Secure Indexes and Data Structures

We now apply the theory to practical privacy-preserving constructions.

7.1 Oblivious Bloom Filters

Bloom filters become oblivious when we hide which bit positions are accessed:

Construction 7.1 (Oblivious Bloom Filter)**.** An *oblivious Bloom filter* for set S consists of:

1. Standard Bloom filter bit array $B[0..m-1]$
2. ORAM or other access-hiding mechanism
3. Query algorithm that accesses bits obliviously

The query for x accesses k bit positions determined by hash functions, but the access pattern is hidden by the ORAM layer.

Proposition 7.2 (Oblivious Bloom Leakage)**.** *An oblivious Bloom filter has:*

- *Access pattern leakage:* $O(k \log m)$ *per query (ORAM overhead)*
- *Result leakage: Bounded by false positive rate* $\alpha = (1 - e^{-kn/m})^k$

Total leakage is the sum of both sources.

7.2 Tuple Encoding for Correlation Hiding

Correlations between queries leak information. Tuple encoding hides them:

Construction 7.3 (Tuple Encoding)**.** For queries that might correlate (e.g., (q_1, q_2) sharing keywords), encode tuples as atomic units:

1. Identify query patterns with frequency above threshold τ

2. Create encoding for each frequent tuple

3. Process tuple as single atomic query

Correlations within tuples are hidden because the tuple is processed atomically.

Theorem 7.4 (Tuple Encoding Privacy)**.** *Tuple encoding eliminates intra-tuple correlation leakage:*

$$I((q_1, q_2); \pi_{tuple}) \leq I((q_1, q_2); \pi_{q_1}, \pi_{q_2}) \tag{35}$$

with equality when tuple frequencies match component product.

Proof. Processing (q_1, q_2) as a single unit means only one trace is produced. Any function of a single trace has at most as much information as the trace itself. Separate processing produces two traces whose joint information exceeds any single trace. ■

The tuple encoding strategy is particularly powerful when combined with uniform encoding. By sizing tuple encodings inversely to frequency, we achieve uniform output distribution even for complex query patterns.

7.3 Oblivious Data Structure Patterns

The principles extend to general data structures:

Definition 7.5 (Oblivious Data Structure)**.** A data structure is *oblivious* if its access patterns are independent of the data stored or operations performed. Common techniques:

1. **Padding**: Access fixed number of elements regardless of operation

2. **Shuffling**: Randomly permute elements periodically

3. **Dummy operations**: Perform fake accesses to mask real ones

4. **ORAM wrapping**: Use oblivious RAM for all storage access

Proposition 7.6 (Oblivious Structure Overhead)**.** *Achieving obliviousness incurs overhead:*

- *Array access:* $O(\log N)$ *(ORAM)*
- *Sorting:* $O(\log^2 N)$ *(oblivious sort)*
- *Binary search:* $O(\log N)$ *(must touch all levels)*
- *Hash table:* $O(\log N)$ *(ORAM for bucket access)*

These overheads are fundamental—Goldreich and Ostrovsky's ORAM lower bound proves $\Omega(\log N)$ is necessary for oblivious memory access.

7.4 Privacy-Accuracy-Performance Trade-off

Real systems navigate a three-way trade-off:

Theorem 7.7 (Privacy-Accuracy-Performance Triangle)**.** *For an oblivious Bernoulli data structure:*

1. ***Privacy*** *(low leakage): Requires uniform encoding + access hiding*
2. ***Accuracy*** *(low error rate): Requires larger encodings + less approximation*
3. ***Performance*** *(low overhead): Requires smaller structures + less oblivious machinery*

Improving any two typically degrades the third.

The framework makes this trade-off explicit. Given requirements on any two dimensions, we can compute the constraint on the third.

8 Related Work

8.1 Oblivious RAM

Oblivious RAM (ORAM), introduced by Goldreich and Ostrovsky [**?**], provides access pattern hiding for memory. Our framework views ORAM as achieving whole-program obliviousness at $O(\log N)$ cost. Path ORAM [**?**] and its variants achieve practical constants. We show how Bernoulli types can reduce ORAM overhead through approximate obliviousness.

8.2 Private Information Retrieval

Private Information Retrieval (PIR), introduced by Chor et al. [**?**], allows querying a database without revealing which item is accessed. Information-theoretic PIR requires multiple servers; computational PIR works with one server but has higher overhead. Our framework views PIR as oblivious point queries, connecting to the broader oblivious type theory.

8.3 Searchable Encryption

Searchable encryption [**? ?**] enables search over encrypted data but typically leaks access patterns and search patterns. The encrypted search literature extensively studies leakage profiles [**?**]. Our leakage decomposition (Theorem 5.3) formalizes this analysis within a broader framework.

8.4 Differential Privacy

Differential privacy [**?**] adds noise to query results to protect individual records. While our framework focuses on access pattern privacy rather than result privacy, there are connections: both use noise/approximation for privacy, both have composition theorems, and both trade accuracy for privacy.

9 Conclusion

We have developed oblivious types as a framework for privacy-preserving computation. The central insight—uniformity equals privacy—transforms privacy from a cryptographic property into an information-theoretic one.

Key contributions include:

- The three-level obliviousness hierarchy (perfect, statistical, computational)
- The random oracle paradox explaining fundamental limits
- Leakage decomposition separating access patterns from result approximation
- Uniform encoding construction achieving privacy through encoding design
- Integration with Bernoulli types providing controlled approximation for privacy

The framework unifies diverse constructions—ORAM, secure indexes, private information retrieval—under one theoretical lens. It provides both understanding of fundamental trade-offs and practical guidance for system design.

Future directions include:

- Tighter bounds on composition leakage
- Adaptive security against adversaries who change strategy
- Integration with differential privacy
- Practical systems achieving near-optimal trade-offs

References