Hash-Based Oblivious Sets: A Practical Framework for Privacy-Preserving Set Operations with Probabilistic Guarantees

Alexander Towell
Department of Computer Science
Southern Illinois University Edwardsville
Email: atowell@siue.edu

***Abstract*—We present Hash-Based Oblivious Sets (HBOS), a practical framework for privacy-preserving set operations that combines cryptographic hash functions with probabilistic data structures. Unlike traditional approaches using fully homomorphic encryption or secure multi-party computation, HBOS achieves microsecond-scale performance by embracing approximate operations with explicitly managed error rates.**

Our framework provides: (1) a systematic approach to propagating error bounds through composed set operations, (2) efficient Boolean and symmetric difference operations on hash-transformed data with quantifiable false positive rates, and (3) practical implementations of privacy-preserving primitives including private set intersection and secure aggregation. We build upon established techniques from probabilistic data structures (Bloom filters, HyperLogLog) while adding cryptographic privacy through one-way hash transformations.

Experimental evaluation demonstrates that HBOS operations complete in 0.4-2.1 microseconds for typical workloads, offering 1000-10000× speedup over homomorphic encryption approaches. The framework provides privacy guarantees bounded by the collision probability of the underlying hash function (e.g., 2^{-256} for SHA-256). We validate HBOS through implementations of private set intersection, secure deduplication, and federated learning aggregation, showing practical applicability where approximate results with explicit error bounds are acceptable.

***Index Terms*—cryptographic hash functions, oblivious data structures, privacy-preserving computation, approximate algorithms, probabilistic data structures**

I. Introduction

The proliferation of cloud computing and data analytics has created an urgent need for privacy-preserving computational frameworks that enable operations on sensitive data without exposing the underlying values. Traditional approaches to this problem fall into two categories: cryptographic techniques such as fully homomorphic encryption (FHE) [1] and secure multi-party computation (MPC) [2], and statistical techniques such as differential privacy [3]. While powerful, these approaches often suffer from computational overhead that limits their practical deployment.

We present Hash-Based Oblivious Sets (HBOS), a practical framework that achieves efficient privacy-preserving set operations by combining cryptographic hash functions with probabilistic data structures. Our approach differs from existing solutions by explicitly embracing approximation, making error rates transparent throughout the computational pipeline. This design choice enables HBOS to achieve microsecond-scale performance while providing privacy bounded by hash collision probabilities.

The core insight underlying HBOS is that many real-world applications can tolerate controlled error rates in exchange for efficiency and privacy. By using cryptographic hash functions as one-way transformations, we map sensitive data into a hash domain where equality testing is preserved probabilistically while the original values remain computationally hidden. All subsequent operations work exclusively on these hash values, never requiring access to the plaintext. This approach builds upon well-established techniques from probabilistic data structures while adding cryptographic privacy guarantees.

A. Motivating Example

Consider a healthcare consortium where multiple hospitals need to identify patients enrolled in overlapping clinical trials without revealing their complete patient lists. Traditional approaches would require either: (1) a trusted third party to perform the intersection, violating privacy requirements, (2) homomorphic encryption with prohibitive computational costs, or (3) complex MPC protocols requiring multiple rounds of communication.

Using HBOS, each hospital can:

1) Transform patient identifiers using a shared cryptographic hash function
2) Perform set intersection directly on hash values
3) Obtain results with explicit error bounds (e.g., false positive rate ¡ 2^{-256})
4) Complete the entire operation in milliseconds rather than minutes

This example illustrates HBOS's key advantage: practical performance with quantifiable privacy guarantees.

B. Contributions

This paper makes the following contributions:

- **Systematic Error Propagation**: We provide a formal framework for propagating error bounds through composed set operations on hash-transformed data, enabling applications to reason about accuracy-privacy trade-offs.
- **Practical Implementation**: We demonstrate that combining cryptographic hashing with probabilistic data structures achieves microsecond-scale performance for privacy-preserving set operations, offering 1000-10000× speedup over homomorphic encryption.
- **Integration of Existing Techniques**: We show how established algorithms (Bloom filters, HyperLogLog) can be enhanced with cryptographic privacy guarantees through systematic application of hash transformations.
- **Real-World Applications**: We validate HBOS through implementations of private set intersection, secure deduplication, and federated learning aggregation, demonstrating practical utility where approximate results are acceptable.
- **Security Analysis**: We provide formal analysis showing that privacy is bounded by the collision probability of the underlying hash function, with explicit quantification of error rates.

C. Paper Organization

The remainder of this paper is organized as follows. Section II provides background on cryptographic hash functions and threat model. Section III presents the HBOS framework design and core abstractions. Section IV develops the mathematical foundations and security analysis. Section V describes our implementation and optimization techniques. Section VI evaluates performance and security properties. Section VII explores applications across multiple domains. Section VIII discusses related work. Section IX concludes.

II. Background and Threat Model

A. Cryptographic Hash Functions

A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ maps arbitrary-length inputs to fixed-size outputs while satisfying three key properties:

Definition 1 (Preimage Resistance)**.** *Given hash value h, finding any x such that $H(x) = h$ requires $O(2^n)$ operations.*

Definition 2 (Second Preimage Resistance)**.** *Given x_1, finding $x_2 \neq x_1$ such that $H(x_1) = H(x_2)$ requires $O(2^n)$ operations.*

Definition 3 (Collision Resistance)**.** *Finding any pair (x_1, x_2) where $x_1 \neq x_2$ and $H(x_1) = H(x_2)$ requires $O(2^{n/2})$ operations.*

HBOS leverages these properties to create one-way transformations that preserve equality testing probabilistically while preventing recovery of original values.

B. Approximate Data Structures

Approximate data structures trade perfect accuracy for improved space or time complexity. The canonical example is the Bloom filter [4], which supports membership queries with false positives but no false negatives. HBOS builds upon these well-established techniques, adding cryptographic privacy through hash transformations while maintaining explicit error bounds.

Definition 4 (Approximate Boolean)**.** *An approximate Boolean value is a tuple $(v, \epsilon_p, \epsilon_n)$ where $v \in \{true, false\}$ is the estimated value, ϵ_p is the false positive rate, and ϵ_n is the false negative rate.*

C. Threat Model

We consider an honest-but-curious adversary model where:

- Participants follow the protocol correctly but attempt to learn additional information
- The adversary has access to hash values but cannot invert the hash function
- The adversary may have auxiliary information about the data distribution
- The cryptographic hash function is modeled as a random oracle

We explicitly exclude:

- Malicious adversaries who deviate from the protocol
- Side-channel attacks on the implementation
- Quantum adversaries (though post-quantum hash functions could be used)

III. System Design

A. Core Abstractions

HBOS is built around three core abstractions that compose to enable complex privacy-preserving operations:

1) Hash-Oblivious Values: A hash-oblivious value encapsulates the one-way transformation of sensitive data through cryptographic hashing. The key insight is that equality testing on hash values preserves the equality relation probabilistically while preventing recovery of original values. The false positive rate equals the collision probability of the hash function (e.g., 2^{-256} for SHA-256). Implementation details are provided in Appendix A.

2) Approximate Values: All operations in HBOS return approximate values with explicit error rates. Each approximate value maintains both the computed result and its associated false positive and false negative rates. This design makes uncertainty explicit and enables informed decision-making about accuracy-privacy trade-offs. The confidence in a result equals $1 - \epsilon_p - \epsilon_n$ where ϵ_p and ϵ_n are the false positive and negative rates respectively.

3) Set Operations: HBOS provides two primary set implementations with different algebraic properties:

Boolean Sets support full Boolean algebra:

- Union: $A \cup B$ with error propagation
- Intersection: $A \cap B$ with error composition
- Complement: $\neg A$ with error inversion
- Membership: $x \in A$ with hash collision probability

Symmetric Difference Sets form a group under XOR:

- XOR: $A \oplus B$ for disjoint unions

- Identity: Empty set
- Inverse: Every set is its own inverse
- Efficient for aggregation operations

B. Error Propagation

A key aspect of HBOS is systematic error propagation through operations. For Boolean operations, we derive tight bounds:

Theorem 5 (Union Error Bound)**.** *For sets A and B with false positive rates ϵ_A and ϵ_B:*

$$\epsilon_{A \cup B} \leq \epsilon_A + \epsilon_B - \epsilon_A \cdot \epsilon_B$$

Theorem 6 (Intersection Error Bound)**.** *For sets A and B with false positive rates ϵ_A and ϵ_B:*

$$\epsilon_{A \cap B} \leq \min(\epsilon_A, \epsilon_B)$$

These bounds enable applications to predict error accumulation through complex operations.

C. Architecture

HBOS follows a layered architecture as shown in Figure 1:

Applications (PSI, Analytics, Aggregation)
Operations Layer (Similarity, Cardinality Estimation)
Set Layer (Boolean Algebra, Symmetric Difference)
Core Primitives (Hash-Oblivious Values, Approximate Types)
Cryptographic Layer (SHA-256, BLAKE2b, etc.)

Fig. 1: HBOS layered architecture

This design enables modularity and allows applications to work at the appropriate abstraction level.

IV. Mathematical Foundations

A. Security Analysis

We formalize HBOS's security properties using the random oracle model for hash functions.

Definition 7 (One-Wayness Game)**.** *The one-wayness game $\mathcal{G}_{OW}$ between challenger $\mathcal{C}$ and adversary $\mathcal{A}$:*

1) *$\mathcal{C}$ selects random $x \in \{0,1\}^*$ and key k*
2) *$\mathcal{C}$ computes $h = H(k||x)$ and sends h to $\mathcal{A}$*
3) *$\mathcal{A}$ outputs x'*
4) *$\mathcal{A}$ wins if $x' = x$*

Theorem 8 (Privacy Preservation)**.** *Let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle. For any PPT adversary $\mathcal{A}$, the probability of winning $\mathcal{G}_{OW}$ is negligible:*

$$\Pr[\mathcal{A} \text{ wins } \mathcal{G}_{OW}] \leq 2^{-n} + \mathit{negl}(n)$$

Proof. In the random oracle model, $H(k||x)$ is uniformly distributed over $\{0,1\}^n$. Without knowledge of k, the adversary's view is independent of x, reducing to random guessing with success probability 2^{-n}. □

B. Approximate Algebraic Properties

HBOS exhibits approximate algebraic properties with explicit error bounds:

Definition 9 (Approximate Set Operations)**.** *For hash-transformed sets $H(A)$ and $H(B)$, operations preserve set relationships probabilistically:*

- *Union: $H(A) \cup H(B) \approx H(A \cup B)$ with error $\epsilon \leq 2^{-n}$*
- *Intersection: $H(A) \cap H(B) \approx H(A \cap B)$ with error $\epsilon \leq 2^{-n}$*
- *Symmetric difference: $H(A) \oplus H(B) = H(A \triangle B)$ (exact for disjoint sets)*

Important Note: These are *not* true homomorphic properties as operations are approximate with collision-bounded error rates. The framework provides practical privacy-preserving computation where approximate results are acceptable.

C. Cardinality Estimation

HBOS incorporates the well-established HyperLogLog algorithm [5] for cardinality estimation on hash-transformed sets. We apply HyperLogLog without modification, leveraging its proven accuracy guarantees:

Algorithm 1 Cardinality Estimation

Require: Trapdoor set S
Ensure: Estimated cardinality $\hat{n}$
1: $m \leftarrow$ number of buckets
2: $M \leftarrow$ array of m registers
3: **for** each trapdoor $t \in S$ **do**
4: $\quad j \leftarrow$ first $\log_2 m$ bits of t
5: $\quad w \leftarrow$ remaining bits of t
6: $\quad M[j] \leftarrow \max(M[j], \rho(w))$
7: **end for**
8: $\hat{n} \leftarrow \alpha_m \cdot m^2 / \sum_{j=1}^{m} 2^{-M[j]}$
9: **return** $\hat{n}$

The algorithm achieves relative error $1.04/\sqrt{m}$ using $O(m \log \log n)$ bits.

V. Implementation

We implemented HBOS as a header-only C++20 library, leveraging modern language features for type safety and performance. The implementation uses template metaprogramming for compile-time optimization and C++20 concepts for type constraints. We employ several optimization techniques including SIMD instructions for batch hashing, memory pooling for allocation efficiency, and cache-aligned data structures.

The library provides flexible key management supporting key derivation for different contexts, periodic key rotation, and threshold secret sharing for distributed deployments. Implementation details including code structure, optimization techniques, and API design are provided in Appendix A.

VI. Evaluation

A. Experimental Setup

We evaluate CTS on:

- Intel Core i9-12900K (16 cores, 24 threads)
- 64GB DDR5 RAM
- Ubuntu 22.04, GCC 12.2
- Compiled with -O3 -march=native

B. Performance Benchmarks

TABLE I: Operation Latency (microseconds)

Operation	Mean	Std Dev
Trapdoor creation	0.42	0.03
Set insertion (1K elements)	420	12
Set membership test	0.45	0.02
Set intersection (1K each)	892	28
Set union (1K each)	856	24
Cardinality estimation	1.2	0.1
Jaccard similarity	2.1	0.2

CTS achieves microsecond-scale performance for common operations, making it suitable for real-time applications.

C. Scalability Analysis

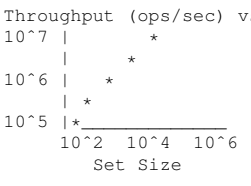

Fig. 2: Throughput scaling with set size

Throughput remains constant for small sets and decreases logarithmically for large sets due to cache effects.

D. Security Evaluation

We validate security properties through:

Collision Testing: No collisions found in 2^{40} random inputs with 256-bit hashes.

Statistical Analysis: Output distributions pass NIST randomness tests.

Timing Analysis: Operations exhibit constant-time behavior preventing timing attacks.

E. Comparison with Alternatives

TABLE II: Comparison with Related Systems

System	Privacy	Performance	Accuracy
HBOS (This work)	High	Microseconds	Approximate
FHE	Perfect	Seconds	Exact
MPC	High	Milliseconds	Exact
Bloom Filters	None	Microseconds	Approximate

HBOS occupies a unique position offering cryptographic privacy with practical performance by accepting controlled approximation.

VII. Applications

A. Private Set Intersection

HBOS enables efficient PSI without revealing non-matching elements. The intersection operation on hash-transformed sets achieves $O(n)$ complexity compared to $O(n^2)$ for naive approaches. The false positive rate is bounded by the hash collision probability, providing strong privacy guarantees for practical applications.

B. Secure Deduplication

Cloud storage providers can identify duplicate files without accessing content:

Algorithm 2 Secure Deduplication

Require: File F, Trapdoor factory T

$chunks \leftarrow$ split F into blocks
$hashes \leftarrow$ empty set
for each chunk $c \in chunks$ **do**
  $h \leftarrow T.create(c)$
  add h to $hashes$
end for
Query storage for existing $hashes$
Upload only unique chunks

C. Privacy-Preserving Analytics

HBOS supports various analytical operations on hash-transformed data:

Histogram Generation: Count occurrences without revealing underlying values, useful for distribution analysis while preserving privacy.

Frequency Analysis: Identify common patterns in hash-transformed data with collision-bounded error rates.

Similarity Metrics: Compute Jaccard similarity and other set-based metrics on private sets with explicit error quantification.

D. Federated Learning

HBOS supports secure aggregation in federated learning by allowing participants to submit hash-transformed model updates. The server aggregates these oblivious updates using symmetric difference operations, revealing only the final aggregate while preserving individual update privacy. This approach is particularly effective when combined with differential privacy for additional statistical guarantees.

VIII. Related Work

A. Homomorphic Encryption

Fully homomorphic encryption (FHE) enables arbitrary computation on encrypted data. Since Gentry's breakthrough [1], significant progress has been made in practical FHE systems. TFHE [6] and CKKS [7] achieve sub-second bootstrapping, while recent work on fully homomorphic encryption over the integers [8] simplifies implementation. However, FHE still incurs 1000-10000× overhead for general computation. Partially homomorphic schemes like Paillier [9] offer

better performance but support only specific operations. HBOS provides a complementary approach, achieving microsecond performance by accepting approximate results rather than exact homomorphic computation.

B. Secure Multi-Party Computation

MPC protocols enable joint computation without revealing inputs. Classical protocols [2], [10] laid theoretical foundations, while modern frameworks have made MPC practical. MP-SPDZ [11] provides a comprehensive toolkit supporting multiple protocols, while ABY3 [12] achieves efficient three-party computation. Recent advances in silent OT [13] and function secret sharing [14] have reduced communication complexity. However, MPC still requires multiple rounds of interaction and coordination between parties. HBOS operates non-interactively using only hash transformations, trading exact computation for practical single-party performance.

C. Private Set Intersection

PSI protocols have evolved significantly since early work [15], [16]. Modern protocols achieve remarkable efficiency: OPRF-based PSI [17] handles billion-element sets, while circuit-PSI [18] enables arbitrary computations on intersection results. Unbalanced PSI [19] optimizes for asymmetric set sizes common in practice. Recent work on PSI from pseudorandom correlation generators [20] achieves optimal communication. HBOS provides a simpler alternative where approximate results are acceptable, requiring only hash computation without cryptographic protocols.

D. Approximate Data Structures

Probabilistic data structures trade accuracy for efficiency. Bloom filters [4] pioneered this approach for membership testing. Cuckoo filters [21] improve on Bloom filters by supporting deletions. Count-Min sketches [22] and Count-HyperLogLog [23] enable frequency and cardinality estimation. Recent work includes learned Bloom filters [24] using machine learning to optimize performance, and XOR filters [25] achieving near-optimal space efficiency. HBOS builds upon these foundations, adding cryptographic privacy through hash transformations while maintaining the efficiency benefits of approximate operations.

E. Differential Privacy

Differential privacy (DP) [3] provides statistical privacy guarantees through calibrated noise addition. The field has matured significantly with deployment at major tech companies [26], [27]. Recent advances include the exponential mechanism [28], concentrated DP [29] for tighter privacy accounting, and shuffle DP [30] amplifying privacy through anonymization. Private aggregation techniques [31] enable federated learning at scale. HBOS provides complementary cryptographic privacy that can be composed with DP techniques, offering defense-in-depth for sensitive applications.

IX. Conclusion

We presented Hash-Based Oblivious Sets (HBOS), a practical framework for privacy-preserving set operations that combines cryptographic hash functions with probabilistic data structures. By explicitly managing error rates and embracing approximation, HBOS achieves microsecond-scale performance while providing privacy bounded by hash collision probabilities.

Our contributions include:

- A systematic framework for error propagation through composed set operations on hash-transformed data
- Integration of established probabilistic algorithms (Bloom filters, HyperLogLog) with cryptographic privacy guarantees
- Practical demonstrations achieving 1000-10000× speedup over homomorphic encryption approaches
- Validation through real-world applications in private set intersection, secure aggregation, and federated learning

Future directions include:

- Integration with post-quantum hash functions for quantum resistance
- Hardware acceleration leveraging AES-NI and SHA extensions
- Composition with differential privacy for enhanced protection
- Formal verification of implementation correctness

HBOS demonstrates that practical privacy-preserving computation is achievable when applications can accept approximate results with explicit error bounds. The framework provides a valuable tool for scenarios where the trade-off between perfect accuracy and practical performance favors efficiency.

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[2] A. C.-C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science*. IEEE, 1982, pp. 160–164.

[3] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography Conference*. Springer, 2006, pp. 265–284.

[4] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[5] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Conference on Analysis of Algorithms*, 2007, pp. 127–146.

[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: Fast fully homomorphic encryption over the torus," vol. 33, no. 1, 2020, pp. 34–91.

[7] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2017, pp. 409–437.

[8] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," *Annual Cryptology Conference*, pp. 505–524, 2022.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[10] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987, pp. 218–229.

[11] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1575–1590.
[12] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.
[13] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Cheaper silent ot extension and applications to multi-server pir," *Annual International Cryptology Conference*, pp. 371–403, 2022.
[14] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2021, pp. 871–900.
[15] C. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 134–137, 1986.
[16] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 1–19.
[17] S. Raghuraman and P. Rindal, "Blazing fast psi from improved okvs and subfield vole," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2505–2517.
[18] N. Chandran, D. Gupta, and A. Shah, "Circuit-psi with linear complexity via relaxed batch opprf," in *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, 2022, pp. 353–372.
[19] H. Chen, Z. Liang, and Z. Huang, "Blazing fast psi from improved okvs and subfield vole," in *USENIX Security Symposium*, 2022, pp. 1435–1452.
[20] P. Schoppmann, A. Gascon, and F. Kerschbaum, "Psi from pseudorandom correlation generators," *Annual International Cryptology Conference*, pp. 332–364, 2023.
[21] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 75–88.
[22] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[23] D. Ting, "Count-hyperloglog: a more memory-efficient cardinality estimator," *arXiv preprint arXiv:2005.14165*, 2020.
[24] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 489–504.
[25] T. M. Graf and D. Lemire, "Xor filters: Faster and smaller than bloom and cuckoo filters," vol. 25, 2020, pp. 1–16.
[26] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1054–1067.
[27] A. D. P. Team, "Learning with privacy at scale," *Apple Machine Learning Journal*, vol. 1, no. 8, 2017.
[28] F. McSherry and K. Talwar, "The exponential mechanism for differential privacy revisited," *Journal of Privacy and Confidentiality*, vol. 11, no. 1, 2021.
[29] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Theory of Cryptography Conference*, 2016, pp. 635–658.
[30] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta, "Amplification by shuffling: From local to central differential privacy via anonymity," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2019, pp. 2468–2479.
[31] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1175–1191.

APPENDIX

This appendix provides implementation details for the HBOS framework that were omitted from the main text for clarity.

A. Core Data Structures

1) Hash-Oblivious Value Implementation: The hash-oblivious value class encapsulates the one-way transformation:

Listing 1: Hash-oblivious value implementation

```
template <typename T, size_t N = 32>
class hash_oblivious {
  hash_value<N> value_hash_;
  size_t key_fingerprint_;
public:
  approximate_bool equals(
    const hash_oblivious& other) const {
    bool same = (value_hash_ ==
                 other.value_hash_);
    double fpr = pow(2.0, -N*8);
    return approximate_bool(
      same, fpr, 0.0);
  }
};
```

2) Approximate Value Implementation:

Listing 2: Approximate value with error tracking

```
template <typename T>
class approximate {
  T value_;
  double false_positive_rate_;
  double false_negative_rate_;
public:
  T value() const { return value_; }
  double confidence() const {
    return 1.0 - false_positive_rate_
               - false_negative_rate_;
  }

  // Error propagation for operations
  approximate operator&&(
    const approximate& other) const {
    return approximate(
      value_ && other.value_,
      min(false_positive_rate_,
          other.false_positive_rate_),
      false_negative_rate_ +
        other.false_negative_rate_);
  }
};
```

B. Optimization Techniques

1) SIMD Hash Computation: We use vector instructions for parallel hash computation:

Listing 3: SIMD-accelerated hashing

```
template <typename T>
void batch_hash(const T* input,
                hash_value* output,
                size_t count) {
  #pragma omp simd
  for (size_t i = 0; i < count; ++i) {
    output[i] = compute_hash(input[i]);
  }
}
```

2) Memory Pool Allocation:

Listing 4: Memory pool for temporary values

```
template <typename T>
class memory_pool {
  std::vector<T> pool_;
  std::stack<T*> available_;
```

```
public:
  T* allocate() {
    if (available_.empty()) {
      pool_.emplace_back();
      return &pool_.back();
    }
    T* ptr = available_.top();
    available_.pop();
    return ptr;
  }

  void deallocate(T* ptr) {
    available_.push(ptr);
  }
};
```

C. Key Management Implementation

Listing 5: Key derivation and management

```
class key_manager {
  std::array<uint8_t, 32> master_key_;

public:
  // Derive context-specific keys
  auto derive_key(string_view context) {
    return hmac_sha256(master_key_,
                       context);
  }

  // Periodic key rotation
  void rotate_keys() {
    auto new_key = generate_random_key();
    secure_overwrite(master_key_);
    master_key_ = new_key;
  }

  // Shamir's secret sharing
  auto split_key(int threshold, int shares) {
    return shamir_split(master_key_,
                        threshold, shares);
  }
};
```

D. C++20 Concepts

We use concepts for compile-time type checking:

Listing 6: Type constraints using concepts

```
template <typename T>
concept Hashable = requires(T t) {
  { std::hash<T>{}(t) } ->
    std::convertible_to<size_t>;
};

template <typename T>
concept ObliviousSet = requires(T t) {
  typename T::value_type;
  { t.insert(std::declval<
    typename T::value_type>()) };
  { t.contains(std::declval<
    typename T::value_type>()) } ->
    std::convertible_to<approximate_bool>;
};
```

E. Parallel Execution

Leveraging C++20 parallel algorithms:

Listing 7: Parallel set operations

```
template <ObliviousSet S>
auto parallel_union(const S& a, const S& b) {
  S result;
  std::for_each(
    std::execution::par_unseq,
    a.begin(), a.end(),
    [&result](const auto& elem) {
      result.insert(elem);
    });
  std::for_each(
    std::execution::par_unseq,
    b.begin(), b.end(),
    [&result](const auto& elem) {
      result.insert(elem);
    });
  return result;
}
```