VIFP: Shape-Hiding Tree Mining on Encrypted Data via Virtual-Interval FP-Growth

Quan Van
Independent Researcher
vanhaminhquan2406@gmail.com

Abstract

Privacy-preserving frequent itemset mining has been studied for more than two decades, mainly through secure candidate counting, horizontally or vertically partitioned protocols, outsourced encrypted support computation, and partial homomorphic outsourcing. However, existing methods do not fully solve the problem of executing tree-based frequent pattern mining, such as FP-Growth, directly over encrypted or secret-shared data without an online decryption-key holder. The core obstacle is structural: FP-Growth derives its efficiency from dynamic prefix trees, recursive conditional pattern bases, variable fan-out, pointer chasing, and data-dependent memory allocation, while fully homomorphic encryption (FHE) and secure multi-party computation (SMPC) are most expensive exactly under hidden branching, hidden memory access, and dynamic data structures.

This paper introduces a new problem called *shape-hiding keyless conditional prefix mining*. The goal is to compute frequent itemsets, or an encrypted representation equivalent to FP-Growth output, while hiding not only the raw transactions but also the evolving tree shape, branch fan-out, conditional database sizes, active prefix identities, support values, and memory-access patterns. We then propose *Virtual-Interval FP-Growth* (VIFP), a new encrypted-tree mining framework that preserves FP-Growth semantics while replacing dynamic FP-tree nodes with static, fixed-width, intervalized array structures. VIFP uses three custom data structures: an Occurrence-Ordered Transaction Tape, a Header-Segmented Posting Array, and Projected Interval Descriptors. These structures turn encrypted pointer chasing into batched scans, stable partitions, and segmented histograms. The framework can be instantiated using additive secret sharing with oblivious sorting and DPF-assisted scatter/gather, or using packed FHE with SIMD support and programmable bootstrapping for threshold tests. We provide formal definitions, a leakage model, algorithmic procedures, correctness arguments, and theoretical complexity analysis showing why VIFP avoids the main bottleneck of direct encrypted FP-tree construction.

Keywords: Privacy-preserving data mining; frequent itemset mining; FP-Growth; secure multi-party computation; fully homomorphic encryption; encrypted data structures; shape-hiding computation; oblivious algorithms.

1 Introduction

Frequent itemset mining is a fundamental problem in data mining. Given a transactional database, the objective is to discover all itemsets whose support is no less than a user-defined minimum support threshold. Classical algorithms such as Apriori, FP-Growth, and Eclat form the foundation of frequent pattern mining [1, 2, 3]. Apriori uses the anti-monotonicity of support to generate and prune candidates level by level. Eclat mines itemsets using vertical transaction identifier sets. FP-Growth avoids explicit candidate generation by compressing the database into an FP-tree and recursively mining conditional pattern bases.

Privacy-preserving frequent itemset mining has also received significant attention. Early secure computation approaches studied horizontally and vertically partitioned association rule mining using secure scalar products, set operations, and secure multi-party computation protocols [4, 5, 6]. More recent outsourced protocols have explored homomorphic encryption, secret sharing, cloud-aided mining,

and encrypted support counting. These works show that frequent itemset mining can be performed while protecting raw transactions. However, most of them are fundamentally *secure candidate-counting systems*. They count supports of candidate itemsets under encryption or shares, rather than executing a full encrypted FP-tree construction and recursive conditional-tree mining process.

Tree-based mining under advanced cryptography remains much less mature. FP-Growth is attractive because it compresses data and avoids candidate explosion in plaintext. However, its main advantages come from data-dependent dynamic structures: prefix-tree insertion, shared path compression, header links, recursive conditional projection, and variable-size memory allocation. These operations are difficult under FHE and SMPC because hidden branches and hidden memory addresses require either expensive circuits, oblivious RAM (ORAM), or many rounds of communication.

Existing encrypted mining approaches often avoid this structural problem. Some protocols outsource only support-counting steps to the encrypted server and leave thresholding, pruning, or tree construction to a client. Others rely on Apriori-like candidate generation, where the secure component only evaluates support queries. These approaches protect raw data but do not provide a fully keyless encrypted FP-Growth engine.

This paper asks a stricter question:

> *Can we execute FP-Growth-like conditional prefix mining over encrypted or secret-shared data while hiding not only transaction values, but also the evolving tree shape, branch fan-out, support values, and memory-allocation pattern, without any online decryption-key holder?*

We answer this question by proposing a new framework called *Virtual-Interval FP-Growth* (VIFP). VIFP preserves the semantics of FP-Growth but eliminates dynamic tree nodes. Instead of materializing an encrypted FP-tree, VIFP transforms the database into fixed-width occurrence records and mines frequent patterns through interval descriptors, batched predecessor extraction, segmented histograms, and stable partitions.

The main contributions are:

- We synthesize the state of privacy-preserving frequent itemset mining and identify the unresolved gap in direct encrypted tree-based mining.
- We define a new problem called *shape-hiding keyless conditional prefix mining*.
- We introduce a formal leakage model that hides raw data, support values, tree shape, branch fan-out, active prefix identities, and memory-access patterns.
- We propose VIFP, a new encrypted mining framework that replaces dynamic FP-tree nodes with static intervalized arrays.
- We design three custom data structures: Occurrence-Ordered Transaction Tape, Header-Segmented Posting Array, and Projected Interval Descriptor.
- We describe SMPC and FHE instantiations using oblivious sorting, DPF-assisted access, SIMD packing, and programmable thresholding.
- We analyze correctness, security intuition, computational complexity, memory complexity, and expected advantages over ORAM-based encrypted tree construction.

2 Related Work

2.1 Frequent Itemset Mining

Apriori introduced the support-based frequent itemset mining framework and the downward-closure principle [1]. FP-Growth improved efficiency by building an FP-tree and mining conditional pattern bases

without explicit candidate generation [2]. Eclat uses vertical transaction identifier sets and computes support by set intersection [3]. These algorithms are not privacy-preserving by themselves, but they provide the algorithmic basis for secure frequent pattern mining.

2.2 Privacy-Preserving Association Rule Mining

Early privacy-preserving association rule mining studied both horizontal and vertical partitioning. In vertically partitioned settings, different parties hold different attributes or items for the same transaction identifiers, and secure scalar products can be used to count joint supports [4]. In horizontally partitioned settings, each party holds a subset of transactions, and secure protocols aggregate local supports without revealing private transactions [5]. Later protocols improved efficiency, reduced leakage, or provided stronger security models [6].

These works are important but largely Apriori-like. They generate candidate itemsets and securely compute their supports. They do not solve encrypted dynamic prefix-tree construction.

2.3 Homomorphic Encryption for Outsourced Mining

Fully homomorphic encryption allows computation on encrypted data without decryption [7]. Since then, many FHE systems have improved batching, bootstrapping, and arithmetic performance [8, 9, 10]. In privacy-preserving data mining, FHE has been used for outsourced support counting and encrypted frequent pattern discovery. However, ciphertext expansion, multiplicative depth, encrypted comparison, and memory bandwidth remain major bottlenecks.

Some FHE-based frequent pattern mining protocols encrypt item-transaction matrices and compute supports homomorphically. These approaches are closer to secure matrix counting than to encrypted FP-Growth. Because comparison and thresholding under FHE are expensive, some protocols send encrypted supports back to the client for decryption and pruning. This violates the keyless mining objective considered in this paper.

2.4 Secure Multi-Party Computation and Dynamic Data Structures

Secure multi-party computation can evaluate functions over private inputs while revealing only the output [11, 12]. Modern MPC systems support arithmetic sharing, Boolean sharing, garbled circuits, oblivious sorting, and private set operations. However, dynamic data structures remain difficult because secret memory addresses require ORAM or equivalent oblivious access techniques [13, 14].

FP-trees are dynamic data structures with data-dependent insertion and traversal. Directly implementing FP-Growth in MPC would require hiding which nodes are accessed, which child branches are created, and how many nodes each conditional tree contains. Generic ORAM can hide accesses but introduces polylogarithmic overhead and high communication cost. This motivates a structure-aware alternative.

2.5 Gap Summary

Existing privacy-preserving frequent itemset mining methods fall into three broad categories:

(1) **Secure candidate counting**: support is computed securely for candidate sets, typically Apriori-style.

(2) **Partial encrypted outsourcing**: some computations, such as initial support counting, are outsourced, but pruning or tree construction remains client-side.

(3) **Generic secure dynamic computation**: dynamic data structures can be emulated with ORAM or RAM-model secure computation, but the cost is too high for FP-Growth-scale mining.

To the best of our survey, no existing work simultaneously provides full FP-Growth semantics, encrypted or secret-shared mining-time execution, no online decryption-key holder, and shape-hiding protection for the evolving tree topology.

3 Problem Formulation

3.1 Transaction Database

Let

$$\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$$

be a finite item universe. A transaction database is denoted by

$$\mathcal{D} = \{T_1, T_2, \ldots, T_n\},$$

where each transaction

$$T_j \subseteq \mathcal{I}.$$

For an itemset $X \subseteq \mathcal{I}$, the supporting transaction set is

$$\Gamma(X) = \{T_j \in \mathcal{D} \mid X \subseteq T_j\}.$$

The support count of X is

$$\sup(X) = |\Gamma(X)|.$$

Given a public support threshold σ, X is frequent if

$$\sup(X) \geq \sigma.$$

The desired plaintext output is

$$\mathcal{F}_\sigma(\mathcal{D}) = \{X \subseteq \mathcal{I} \mid \sup(X) \geq \sigma\}.$$

3.2 Encrypted and Shared Representations

In the FHE setting, the database is stored as ciphertexts:

$$[\![\mathcal{D}]\!].$$

The server has public evaluation keys but does not have the secret decryption key.

In the SMPC setting, the database is secret-shared:

$$\langle \mathcal{D} \rangle.$$

No individual computing party has plaintext access to $\mathcal{D}$.

This paper focuses on protocols that compute either:

$$[\![\mathcal{F}_\sigma(\mathcal{D})]\!]$$

in the FHE setting, or

$$\langle \mathcal{F}_\sigma(\mathcal{D}) \rangle$$

in the SMPC setting.

3.3 Shape-Hiding Leakage Model

Classical privacy-preserving mining usually protects item values and transaction contents. However, tree-based mining can leak information through the shape of the computation itself. We therefore define a stricter leakage model.

Definition 1 (Public Leakage Function)**.** *The protocol may reveal only:*

$$\mathcal{L}(\mathcal{D}) = \{n,\ |\mathcal{I}|,\ \ell_{\max},\ \sigma,\ B_1, \ldots, B_h\},$$

where n is the number of transactions, $|\mathcal{I}|$ is the item universe size, $\ell_{\max}$ is a public upper bound on transaction length, and B_d is a public capacity bound on the number of level-d protocol states.

Everything else must remain hidden, including:

(i) active prefix identities,

(ii) intermediate support counts,

(iii) branch fan-out,

(iv) FP-tree shape,

(v) conditional pattern base sizes,

(vi) node allocation patterns,

(vii) memory-access patterns,

(viii) identities of pruned branches.

Definition 2 (Shape-Hiding Keyless Conditional Prefix Mining)**.** *Given an encrypted or secret-shared transaction database $\mathcal{D}$, a public support threshold σ, and a public leakage function $\mathcal{L}$, the shape-hiding keyless conditional prefix mining problem is to compute an encrypted or secret-shared representation of $\mathcal{F}_\sigma(\mathcal{D})$ such that:*

$$\textit{Correctness:}\ \mathrm{Dec}([\![\mathcal{F}_\sigma(\mathcal{D})]\!]) = \mathcal{F}_\sigma(\mathcal{D}),$$

or equivalently the opened SMPC output equals $\mathcal{F}_\sigma(\mathcal{D})$, and the adversary learns no information beyond $\mathcal{L}(\mathcal{D})$.

Remark 1. *This problem is stricter than ordinary privacy-preserving frequent itemset mining because it protects not only input transactions but also the mining-time control plane.*

4 Technical Bottlenecks in Direct Encrypted FP-Growth

4.1 Dynamic Node Allocation

An FP-tree grows dynamically as transactions are inserted. Each insertion depends on the current transaction prefix and the existing tree structure. In plaintext, this is efficient. Under encryption or secret sharing, the identity of the accessed node and whether a child node is created are sensitive. Hiding these events requires oblivious allocation or padding all possible nodes, both of which are expensive.

4.2 Pointer Chasing

FP-Growth relies on parent pointers, child pointers, and header links. Encrypted pointer chasing requires hiding memory addresses. Generic ORAM can hide the access sequence, but each logical access becomes multiple physical accesses. In a recursive FP-Growth procedure, this cost compounds across many conditional trees.

4.3 Encrypted Comparison and Thresholding

Support counts must be compared with σ. Comparisons are expensive in FHE because they require high-depth Boolean or arithmetic circuits. In SMPC, comparisons require interactive protocols. Many outsourced systems avoid this by letting the client decrypt supports and decide pruning, but this violates keyless mining.

4.4 Conditional Tree Recursion

FP-Growth recursively constructs conditional pattern bases and conditional FP-trees. The size and shape of each conditional structure depend on the data. Therefore, even if raw items are encrypted, the sizes of conditional databases and branch fan-outs may leak sensitive information.

4.5 Ciphertext Expansion and Memory Bandwidth

FHE ciphertexts are much larger than plaintext values. A direct encrypted FP-tree with encrypted nodes, counters, pointers, and header links would multiply the memory footprint dramatically. The system bottleneck becomes memory movement and bootstrapping rather than arithmetic alone.

5 Virtual-Interval FP-Growth

5.1 Design Philosophy

VIFP is based on the following observation:

> FP-Growth does not fundamentally require heap-allocated tree nodes. It requires an order-preserving representation of transaction prefixes, conditional pattern bases, and support histograms.

Therefore, VIFP replaces dynamic FP-tree nodes with static intervalized arrays. The framework preserves FP-Growth semantics but changes the computational representation from a dynamic tree into fixed-width records and contiguous intervals.

The key transformation is:

$$\text{dynamic encrypted FP-tree} \quad \longrightarrow \quad \text{static encrypted occurrence tape + projected intervals.}$$

This eliminates hidden allocation and replaces pointer traversal with batched scans and segmented reductions.

5.2 Data Structure 1: Occurrence-Ordered Transaction Tape

Definition 3 (Occurrence Record)**.** *An occurrence record is a fixed-width tuple:*

$$r = \langle tid,\ pos,\ item,\ pred,\ live \rangle,$$

where:

- tid *is the transaction identifier,*
- pos *is the position of the item in the globally ordered transaction,*
- $item$ *is the item identifier,*
- $pred$ *is the predecessor position in the same transaction path,*

- $live$ *is an activity bit indicating whether the record participates in the current projection.*

Definition 4 (Occurrence-Ordered Transaction Tape)**.** *The Occurrence-Ordered Transaction Tape, denoted by* OTT*, is the flattened sequence of all occurrence records after each transaction has been sorted by a public global item order.*

Let

$$N_{occ} = \sum_{j=1}^{n} |T_j|$$

be the number of item occurrences. Then OTT contains exactly N_{occ} fixed-width records.

Remark 2. OTT *avoids one ciphertext per item-transaction matrix cell. It scales with the number of observed item occurrences rather than* $n|\mathcal{I}|$.

5.3 Data Structure 2: Header-Segmented Posting Array

Definition 5 (Header-Segmented Posting Array)**.** *The Header-Segmented Posting Array, denoted by* HPA*, is obtained by stably sorting* OTT *by item identifier. For each item* i*, all records corresponding to* i *form a contiguous interval:*

$$\mathrm{HPA}[i] = [s_i, e_i).$$

HPA replaces the FP-tree header table and header links. Accessing all occurrences of item i becomes an interval scan rather than linked-list traversal.

5.4 Data Structure 3: Projected Interval Descriptor

Definition 6 (Projected Interval Descriptor)**.** *A Projected Interval Descriptor for prefix* α *is:*

$$\mathrm{PID}(\alpha) = \langle id_\alpha,\ depth_\alpha,\ S_\alpha,\ E_\alpha,\ cap_\alpha \rangle,$$

where:

- id_α *is an encrypted or secret-shared prefix identifier,*
- $depth_\alpha$ *is the prefix length,*
- $[S_\alpha, E_\alpha)$ *is a contiguous interval in a projected occurrence buffer,*
- cap_α *is a public capacity bound assigned to this state.*

A PID is a virtual FP-tree state. It does not allocate child nodes. Its children are created by stable partitioning its projected buffer.

6 Cryptographic Instantiations

6.1 SMPC Instantiation

The most practical first implementation of VIFP is a three-party additive-secret-sharing protocol in the semi-honest model. Data values are represented as shares:

$$x = x_1 + x_2 + x_3 \pmod{2^k}.$$

The protocol uses:

- oblivious sorting for canonicalization and header segmentation,

- additive sharing for counters and item identifiers,
- batched secure equality tests for item grouping,
- secure comparison for threshold tests,
- DPF-assisted scatter/gather for projected interval construction,
- stable oblivious partition for child interval formation.

The SMPC instantiation is communication-heavy but avoids FHE ciphertext expansion and bootstrapping.

6.2 FHE Instantiation

The FHE instantiation uses packed ciphertexts. Multiple occurrence records or support counters are packed into one ciphertext using SIMD batching. Support histograms are computed through batched arithmetic. Thresholding can be implemented using programmable bootstrapping or lookup-table evaluation in schemes such as TFHE-style Boolean/arithmetic hybrids.

The FHE version is less communication-intensive but may suffer from:

- ciphertext expansion,
- expensive comparisons,
- slot-rotation overhead,
- bootstrapping cost,
- memory bandwidth pressure.

Therefore, this paper treats SMPC as the recommended first implementation and FHE as a compatible but more expensive instantiation.

7 VIFP Algorithm

7.1 Overview

VIFP has five phases:

(1) Secure singleton support computation.

(2) Public global item ordering.

(3) Secure canonicalization and flattening into OTT.

(4) Stable header segmentation into HPA.

(5) Recursive virtual-interval mining.

7.2 Global Frequency Order

The protocol first computes singleton supports:

$$c_i = \sup(\{i\}).$$

Items with

$$c_i < \sigma$$

are marked inactive. The remaining frequent singleton items are ordered by descending support. This produces a public global item order π.

Remark 3. *Revealing the global order is included in the public leakage function. A stricter version can hide the order using oblivious sorting and padded item domains, but this increases cost.*

7.3 Virtual Mining Logic

For an item i, the initial projected interval is:

$$\mathrm{PID}(i) = \mathrm{HPA}[i].$$

Given a prefix α, VIFP mines extensions by:

(1) reading all occurrence records in $\mathrm{PID}(\alpha)$,

(2) performing batched predecessor extraction,

(3) constructing the conditional pattern base CPB_α,

(4) computing a segmented histogram of items in CPB_α,

(5) applying secure threshold tests,

(6) stably partitioning CPB_α into child intervals,

(7) recursively mining each child interval.

Algorithm 1 VIFP: Virtual-Interval FP-Growth

Require: Encrypted or secret-shared database $[\![\mathcal{D}]\!]$ or $\langle\mathcal{D}\rangle$, minimum support σ
Ensure: Encrypted or secret-shared frequent itemset representation
1: $[\![c]\!] \leftarrow$ SECURESINGLETONHISTOGRAM$(\mathcal{D})$
2: $\pi \leftarrow$ PUBLICFREQUENCYORDER$([\![c]\!], \sigma)$
3: $[\![\mathrm{OTT}]\!] \leftarrow$ SECURECANONICALIZEANDFLATTEN$(\mathcal{D}, \pi)$
4: $[\![\mathrm{HPA}]\!] \leftarrow$ STABLESORTBYITEM$([\![\mathrm{OTT}]\!])$
5: $\mathcal{P} \leftarrow$ BUILDHEADERINTERVALS$([\![\mathrm{HPA}]\!])$
6: **for** each frequent item i in reverse order of π **do**
7: $\quad \mathrm{PID}(i) \leftarrow \mathcal{P}[i]$
8: $\quad$ MINEINTERVAL$(\mathrm{PID}(i), \{i\})$
9: **end for**

Algorithm 2 MINEINTERVAL

Require: Projected interval descriptor $\mathrm{PID}(\alpha)$, encrypted or shared prefix α
Ensure: Frequent extensions of α

```
[[CPB_α]] ← BATCHEDPREDECESSORFETCH(PID(α))
[[h]] ← SEGMENTEDHISTOGRAM([[CPB_α]])
[[alive]] ← SECUREGEQBATCH([[h]], σ)
EMIT all encrypted/shared extensions x ∪ α where [[alive_x]] = 1
[[children]] ← STABLEPARTITIONBYITEM([[CPB_α]], [[alive]])
for each public capacity slot j in [[children]] do
    PID_j ← BUILDPID([[children]], j)
    MINEINTERVAL(PID_j, α ∪ x_j)
end for
```

7.4 Batched Predecessor Fetch

In plaintext FP-Growth, a conditional pattern base is obtained by following parent pointers from an item occurrence to the root. VIFP replaces this with a batched predecessor fetch.

For each occurrence record

$$r = \langle tid, pos, item, pred, live \rangle,$$

the predecessor is the record in the same transaction with position $pred$. The protocol fetches all predecessors in a batch using oblivious gather or stable self-join on (tid, pos).

Repeated predecessor extraction reconstructs the prefix path in an intervalized form.

7.5 Segmented Histogram

Given a conditional pattern base CPB_α, VIFP computes item supports using a segmented histogram:

$$h_x = |\{r \in \mathrm{CPB}_\alpha \mid r.item = x\}|\,.$$

The threshold test is:

$$alive_x = [h_x \geq \sigma].$$

In SMPC, this is performed by secure comparison. In FHE, it can be performed by batched comparison circuits or programmable bootstrapping.

7.6 Stable Partitioning

The surviving records are stably partitioned by item. Each partition becomes a child interval:

$$\mathrm{PID}(\alpha \cup x) = [S_{\alpha,x}, E_{\alpha,x}).$$

No dynamic child node is allocated. The child state is only an interval descriptor.

8 Correctness Analysis

Theorem 1 (Semantic Equivalence to FP-Growth)**.** *For every prefix α, the conditional pattern base produced by* BATCHEDPREDECESSORFETCH *is equivalent to the conditional pattern base that plaintext FP-Growth would derive from an FP-tree.*

Proof. In an FP-tree, the conditional pattern base of α consists of all prefix paths ending at occurrences of α. In VIFP, every transaction is stored as an ordered path in OTT, and each occurrence record stores its predecessor position. Therefore, repeated predecessor extraction from the occurrences in $\mathrm{PID}(\alpha)$ reconstructs exactly the same prefix paths. Since stable ordering preserves transaction path order, the resulting conditional pattern base is equivalent to the FP-Growth conditional pattern base. □

Theorem 2 (Correct Support Counting)**.** *For every prefix α and candidate extension x,* SEGMENTEDHISTOGRAM *computes the same support count for $x \cup \alpha$ as plaintext FP-Growth.*

Proof. The conditional pattern base CPB_α contains exactly the prefix-path occurrences associated with α. Counting the number of transactions in which x appears in this conditional base is exactly the support of $x \cup \alpha$. Since the segmented histogram counts these occurrences by item, it produces the same support values as plaintext FP-Growth. □

Theorem 3 (Completeness)**.** *VIFP outputs every frequent itemset X with $\sup(X) \geq \sigma$.*

Proof. FP-Growth recursively enumerates all frequent suffix-prefix combinations by mining conditional pattern bases. By semantic equivalence, VIFP constructs the same conditional pattern bases. By correct support counting, VIFP applies the same support threshold test. Therefore, every frequent itemset that would be discovered by FP-Growth is also emitted by VIFP. □

Theorem 4 (Soundness)**.** *Every itemset emitted by VIFP is frequent.*

Proof. VIFP emits an extension $x \cup \alpha$ only when the secure threshold test confirms:

$$\sup(x \cup \alpha) \geq \sigma.$$

By correctness of segmented support counting, this support value is the true support. Therefore, every emitted itemset is frequent. □

9 Security Discussion

9.1 Security Goal

The protocol aims to reveal only:

$$\mathcal{L}(\mathcal{D}) = \{n,\ |\mathcal{I}|,\ \ell_{\max},\ \sigma,\ B_1, \ldots, B_h\}.$$

The adversary should not learn active prefixes, exact support values, conditional tree sizes, branch fan-out, or hidden memory accesses.

9.2 Shape Hiding

VIFP hides shape by avoiding dynamic tree allocation. Instead of creating one node per data-dependent prefix, it allocates public-capacity buffers and interval descriptors. The number of capacity slots per level is public. Whether a slot is active is represented by encrypted or secret-shared flags.

Thus, the physical computation follows a padded public schedule, while logical activity remains private.

9.3 Memory-Access Hiding

VIFP replaces pointer chasing with:

- oblivious stable sorting,
- batched predecessor self-join,
- segmented histogram,
- stable partition.

These operations can be implemented with data-oblivious access patterns. Therefore, memory access depends only on public buffer sizes, not on private transaction contents.

9.4 Keyless Mining

In the FHE setting, all support counting and thresholding are evaluated using public evaluation keys. In the SMPC setting, all intermediate values remain secret-shared. No online decryption-key holder is required during mining-time pruning or recursive state construction.

10 Complexity Analysis

Let

$$N_{occ} = \sum_{j=1}^{n} |T_j|$$

be the number of item occurrences, and let f be the number of frequent singleton items.

10.1 Preprocessing Cost

Singleton support computation costs:

$$O(N_{occ})$$

secure additions or equivalent histogram operations.

Canonicalization and stable sorting cost:

$$O(N_{occ} \log N_{occ})$$

under comparison-based oblivious sorting. If radix sorting over bounded item identifiers is used, the cost can be reduced to:

$$O(N_{occ} \log |\mathcal{I}|)$$

or near-linear in practical MPC implementations.

10.2 Recursive Mining Cost

Let $\mathcal{S}$ be the set of recursive projected states. Let M_α be the size of the projected interval for state α. For each state:

$$O(M_\alpha)$$

work is required for predecessor extraction,

$$O(M_\alpha)$$

work is required for segmented histogramming, and

$$O(f_\alpha)$$

secure threshold tests are needed, where f_α is the number of candidate extensions in state α.

Thus, the total work is:

$$O\left(N_{occ}\log N_{occ} + \sum_{\alpha\in\mathcal{S}}(2M_\alpha + f_\alpha)\right).$$

This is output-sensitive, like plaintext FP-Growth. The worst case remains exponential in f, because frequent itemset mining itself can have exponentially many outputs.

10.3 Memory Complexity

The Occurrence-Ordered Transaction Tape requires:

$$O(N_{occ})$$

fixed-width records.

The Header-Segmented Posting Array also requires:

$$O(N_{occ})$$

records.

Projected interval descriptors require:

$$O(|\mathcal{S}|)$$

small fixed-width descriptors.

Therefore, the total memory is:

$$O(N_{occ} + |\mathcal{S}|),$$

excluding cryptographic expansion factors.

10.4 Comparison with Direct Encrypted FP-Trees

A direct encrypted FP-tree requires encrypted nodes, encrypted counters, encrypted parent pointers, encrypted child pointers, and hidden allocation metadata. If ORAM is used, each logical pointer access incurs polylogarithmic overhead:

$$O(\text{polylog}\, N)$$

per access.

In contrast, VIFP uses batched scans and stable partitions. Its inner loop avoids hidden random access and dynamic allocation. Therefore, VIFP removes the dominant cryptographic overhead of encrypted pointer-based tree mining.

11 Experimental Evaluation

11.1 Baselines

The implementation evaluates VIFP against plaintext frequent-itemset baselines and against two cryptographic execution profiles. The current implementation executes the VIFP logical plan in plaintext while collecting SMPC and FHE proxy metrics for the cryptographic operations described in the paper. This separates semantic correctness and algorithmic work from backend-specific cryptographic engineering.

The compared algorithms are:

- VIFP-Plain, the direct virtual-interval execution.

- VIFP-SMPC, the same logical execution with secret-sharing communication proxies.
- VIFP-FHE, the same logical execution with ciphertext-footprint and bootstrapping proxies.
- FP-Growth, Eclat, Apriori, and FPmax as plaintext frequent-itemset baselines.

11.2 Datasets

The benchmark uses seven transactional datasets from the project repository: Mushroom, Chess, FoodmartFIM, Retail, SYN-Sparse, SYN-Dense, and SYN-Long. Each dataset is evaluated at three support thresholds selected to produce a mix of sparse, dense, and low-output regimes. The resulting benchmark contains 147 runs: seven datasets, seven algorithms or execution profiles, and three support thresholds per dataset. All 147 runs completed successfully.

11.3 Metrics

The benchmark reports:

- runtime,
- peak RAM,
- number of emitted frequent itemsets,
- occurrence tape records,
- projected interval states,
- conditional pattern-base records,
- number of secure comparisons,
- number of oblivious sort operations,
- stable partition count,
- estimated SMPC communication volume,
- estimated FHE ciphertext memory footprint,
- estimated number of FHE bootstrapping operations.

11.4 Implementation and Reproducibility

The VIFP implementation follows Algorithm 1 and Algorithm 2. The executable `vifp_miner` first computes singleton supports, builds the public frequency order, canonicalizes transactions into an Occurrence-Ordered Transaction Tape, derives the Header-Segmented Posting Array, and then recursively mines Projected Interval Descriptors. The implementation intentionally reports only aggregate statistics: it does not print or store the mined pattern contents in the benchmark output. This matches the paper's emphasis on shape-hiding execution statistics rather than revealing pattern identities during protocol evaluation.

The benchmark was run with three support thresholds for each dataset and seven algorithms or execution profiles. The final result matrix contains $7 \times 7 \times 3 = 147$ runs. Every run completed successfully, with no failures, limits, or timeouts. The raw summary is stored under the `results/vifp_compare_full` directory, together with the generated figures.

Table 1: Benchmark completion summary.

Group	Datasets	Algorithms / Modes	Thresholds	Runs
Full benchmark	7	7	3	147
Successful runs	–	–	–	147
Failed / limited / timed-out runs	–	–	–	0

11.5 Semantic Validation

The first experimental question is whether the virtual-interval representation preserves FP-Growth semantics. For every dataset-threshold pair, VIFP-Plain, VIFP-SMPC, and VIFP-FHE produced the same frequent-itemset count as FP-Growth. Across all 21 dataset-threshold configurations, the number of mismatches was zero. This result supports the central claim of the paper: VIFP changes the computational representation from dynamic trees to intervalized arrays without changing the frequent-pattern semantics.

Table 2: Semantic agreement between VIFP and FP-Growth.

Comparison	Dataset-threshold cases	Count mismatches
VIFP-Plain vs FP-Growth	21	0
VIFP-SMPC vs FP-Growth	21	0
VIFP-FHE vs FP-Growth	21	0

11.6 Global Results

Figure 1 summarizes median runtime, output size, RAM, and tail runtime across all completed runs. VIFP-Plain, VIFP-SMPC, and VIFP-FHE emit the same itemset counts because they share the same virtual-interval mining semantics; the SMPC and FHE variants differ in the reported cryptographic proxy metrics.

Table 3 gives the aggregate numerical results. Plaintext FP-Growth is faster than VIFP on several datasets, which is expected: FP-Growth is allowed to use data-dependent pointers, branch fan-out, and dynamic memory allocation. VIFP deliberately pays additional plaintext overhead to expose a computation plan that can be padded, scanned, partitioned, and evaluated under shape-hiding secure computation. Thus, the relevant result is not that VIFP beats plaintext FP-Growth, but that it preserves FP-Growth output while replacing hidden pointer chasing with auditable interval operations.

Table 3: Global aggregate performance across all completed runs.

Algorithm / Mode	Runs	Median runtime (s)	90th-percentile runtime (s)	Median peak RAM (MB)
VIFP-Plain	21	0.002168	0.148965	4.847656
VIFP-SMPC proxy	21	0.002072	0.138680	4.816406
VIFP-FHE proxy	21	0.002322	0.163049	4.800781
FP-Growth	21	0.000843	0.023487	3.640000
Eclat	21	0.003235	0.044771	4.500000
Apriori	21	0.011221	0.210215	3.700000
FPmax	21	0.000698	0.015523	3.510000

Figure 2 gives dataset-level views. This is important because VIFP's cost is driven by occurrence records, conditional pattern-base scans, and the number of projected interval states, which vary strongly across dense and sparse datasets.

Table 4 shows the median runtime and median output size per dataset. The most demanding dataset in this benchmark is Chess: VIFP-Plain has a median runtime of 1.150004 seconds and a median output size of 8,227 frequent itemsets. Retail and Mushroom are moderate-output cases, with median VIFP-Plain

runtimes of 0.050283 seconds and 0.022318 seconds, respectively. FoodmartFIM and the three synthetic datasets are low-output settings under the selected thresholds; they are useful for showing the fixed overhead of the occurrence-tape and interval schedule.

Table 4: Dataset-level median runtime and output.

Dataset	VIFP-Plain median runtime (s)	FP-Growth median runtime (s)	Median output itemsets
Chess	1.150004	0.023487	8227
FoodmartFIM	0.000161	0.000099	0
Mushroom	0.022318	0.000859	51
Retail	0.050283	0.010222	55
SYN-Dense	0.001250	0.000552	0
SYN-Long	0.002168	0.000876	0
SYN-Sparse	0.000382	0.000322	0

11.7 Virtual-Interval Structure

Figure 3 visualizes the VIFP-specific data structures and work units: occurrence records in the Occurrence-Ordered Transaction Tape, projected interval states, and conditional pattern-base scans. These metrics are the implementation-level counterpart of the complexity terms N_{occ}, $|\mathcal{S}|$, and $\sum_\alpha M_\alpha$.

The largest observed VIFP run used 236,563 occurrence records, 48,731 projected interval states, and 121,021,939 conditional pattern-base record visits. These numbers explain the overhead visible on dense outputs such as Chess: VIFP does not rely on cheap pointer traversal. It materializes the secure-computation work as explicit scans and partitions. This is precisely the intended trade-off. The algorithm replaces hidden memory access with a regular work schedule whose shape can be bounded by public capacities.

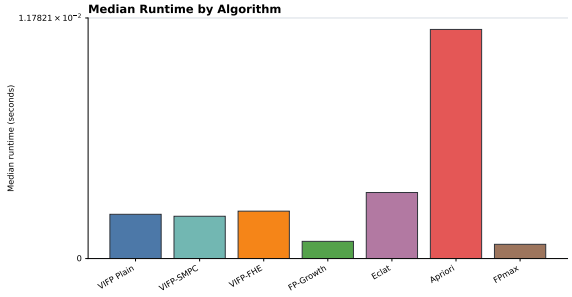
Table 5: Observed VIFP structural work range.

Metric	Median	Maximum
Occurrence records	0	236563
Projected interval states	0	48731
Conditional pattern-base records scanned	0	121021939
Secure threshold comparisons	0	50303
Stable partitions	0	24345
Oblivious sorts	2	2

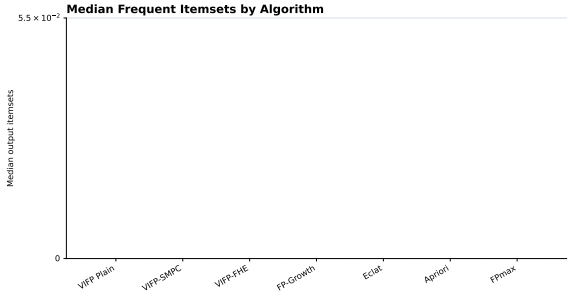
11.8 Cryptographic Proxy Metrics

Figure 4 reports the cryptographic proxy metrics collected by the implementation. Secure comparisons correspond to threshold tests in SEGMENTEDHISTOGRAM. Communication is estimated for the additive secret-sharing profile. Ciphertext footprint and bootstrapping counts are estimated for the packed FHE profile.

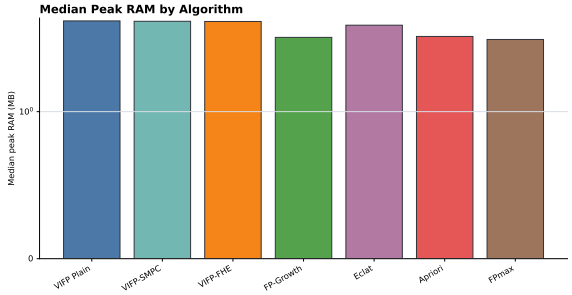
The proxy metrics highlight why VIFP is primarily an SMPC-oriented design. In the largest observed run, the SMPC communication proxy reaches approximately 9.31×10^{10} bytes, while the FHE ciphertext-footprint proxy reaches approximately 1.59×10^{13} bytes and 50,303 bootstrapping-equivalent threshold operations. These figures are not presented as optimized cryptographic measurements; rather, they quantify the secure-computation pressure induced by exact FP-Growth-equivalent mining. They support the design recommendation made earlier: VIFP is best suited first to secret-sharing systems with efficient oblivious sorting and batched segmented reductions, while a practical FHE backend would require careful packing, threshold batching, and bootstrapping optimization.



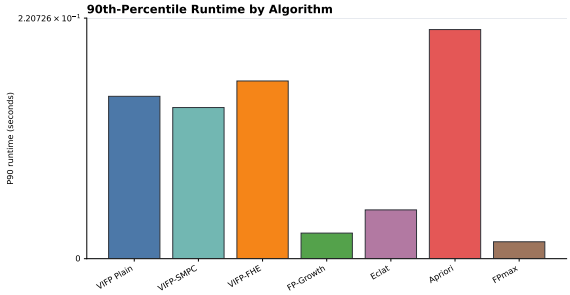
(a) Median runtime.



(b) Median itemsets.

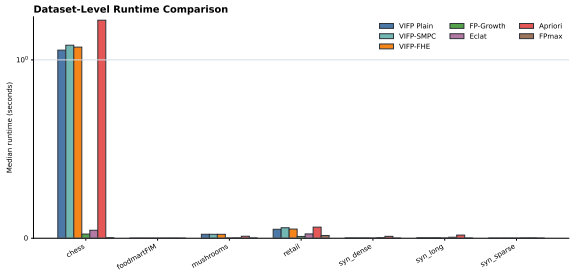


(c) Median peak RAM.

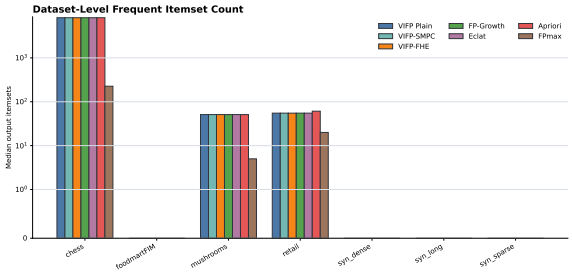


(d) 90th-percentile runtime.

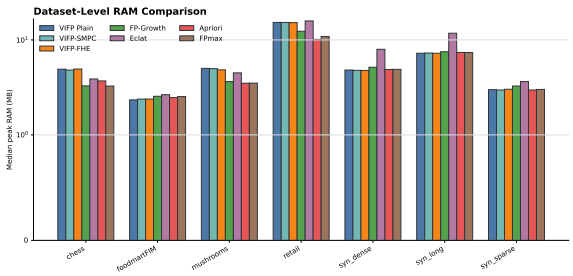
Figure 1: Global VIFP comparison against plaintext frequent-itemset baselines.



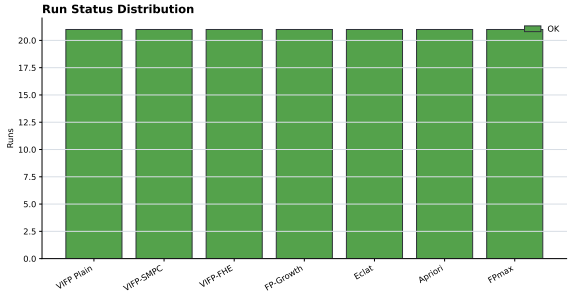
(a) Runtime by dataset.



(b) Itemsets by dataset.



(c) Peak RAM by dataset.



(d) Run status.

Figure 2: Dataset-level benchmark behavior and completion status.

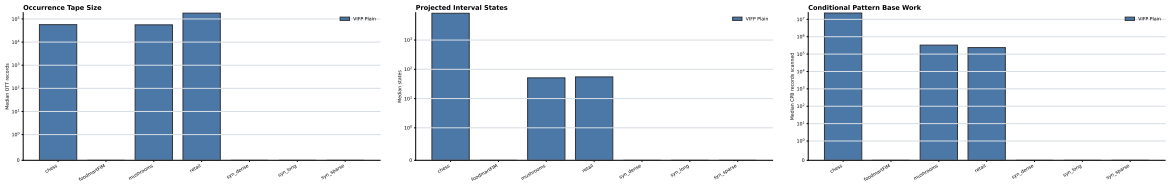
Table 6: Maximum observed cryptographic proxy costs.

Proxy metric	Median	Maximum
SMPC communication estimate (bytes)	0	93096731904
FHE ciphertext footprint estimate (bytes)	0	15882335649792
FHE bootstrap-equivalent threshold tests	0	50303

11.9 Threshold Sensitivity

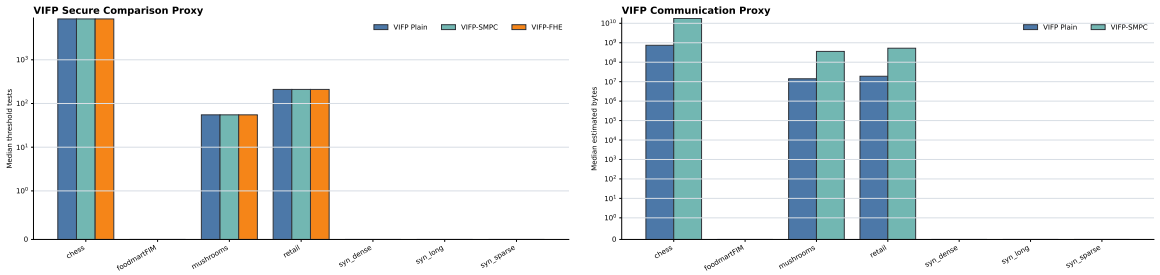
Figure 5 and Figure 6 show runtime sensitivity as the minimum support threshold changes. The x-axis is inverted so stricter thresholds appear on the left and lower thresholds, which usually produce more work and more output, appear on the right.

The threshold curves show the expected behavior: when support decreases, more singletons survive, more projected states are activated, and more conditional pattern-base records must be scanned. This effect is most visible on Chess and Mushroom, where the lower support thresholds produce nontrivial frequent-itemset output. Low-output settings remain near the fixed preprocessing cost, because the occurrence tape and header segmentation are constructed but the recursive interval expansion is shallow.

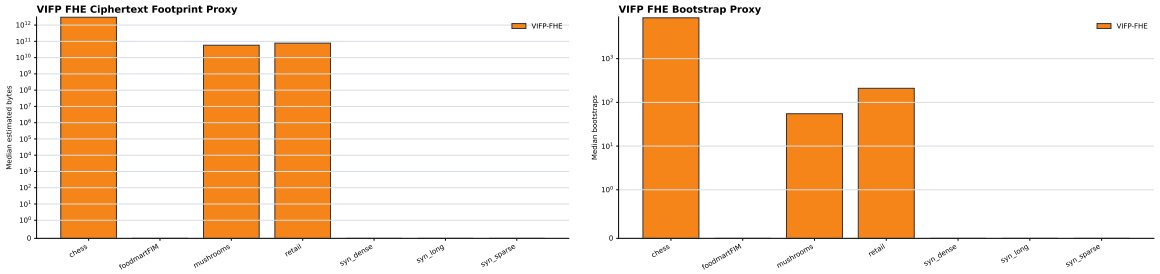


(a) Occurrence tape size. (b) Projected states. (c) CPB work.

Figure 3: VIFP structural work metrics.

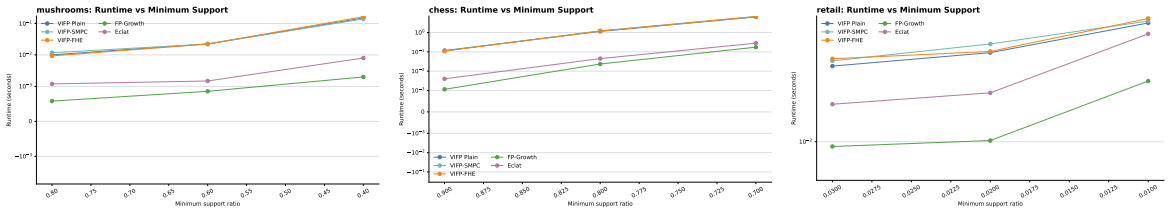


(a) Secure comparison proxy. (b) SMPC communication proxy.



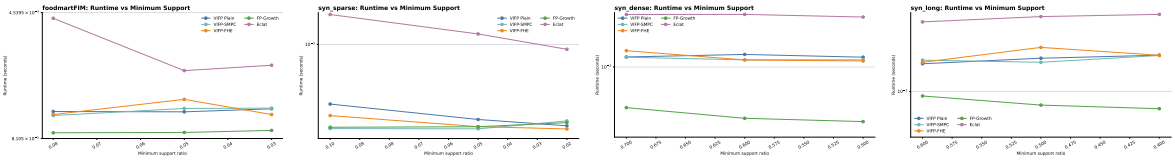
(c) FHE ciphertext proxy. (d) FHE bootstrap proxy.

Figure 4: Cryptographic cost proxies for VIFP-SMPC and VIFP-FHE.



(a) Mushroom. (b) Chess. (c) Retail.

Figure 5: Runtime sensitivity on real datasets.



(a) FoodmartFIM. (b) SYN-Sparse. (c) SYN-Dense. (d) SYN-Long.

Figure 6: Runtime sensitivity on FoodmartFIM and synthetic datasets.

12 Discussion

VIFP is not a minor optimization of FP-Growth. It changes the secure computation object. Instead of encrypting a dynamic tree, it virtualizes the tree as static intervals over occurrence records. This allows the protocol to preserve FP-Growth semantics while avoiding encrypted pointer chasing and hidden allocation.

The framework is especially suitable for SMPC because segmented histograms, stable partitions, and batched predecessor extraction can be implemented efficiently with secret sharing and oblivious array operations. The FHE version is conceptually compatible but likely slower due to bootstrapping and slot movement costs.

The main limitation is that VIFP does not remove the inherent worst-case combinatorial explosion of frequent itemset mining. If the frequent itemset output is exponentially large, any exact protocol must account for that. VIFP addresses the cryptographic overhead of tree construction, not the intrinsic hardness of the mining task.

13 Threats to Validity

The proposed framework is theoretical and requires implementation for empirical validation. The SMPC version assumes a semi-honest threat model. A malicious-secure version would require MACed shares, zero-knowledge checks, or consistency proofs for sorting, histogramming, and interval construction.

The leakage model allows public capacity bounds. If these bounds are too tight, overflow may occur. If they are too loose, computation becomes inefficient. Designing adaptive but shape-hiding capacity schedules remains an open problem.

The FHE instantiation may be impractical for large datasets unless batching and programmable thresholding are carefully optimized.

14 Conclusion

This paper introduced the problem of shape-hiding keyless conditional prefix mining and proposed VIFP, a Virtual-Interval FP-Growth framework for encrypted or secret-shared frequent itemset mining. The main insight is that FP-Growth semantics can be preserved without materializing a dynamic encrypted FP-tree. By replacing tree nodes with occurrence tapes, header-segmented posting arrays, and projected interval descriptors, VIFP transforms hidden pointer chasing into batched scans, segmented histograms, and stable partitions.

The proposed framework hides not only raw transactions but also tree shape, branch fan-out, active prefixes, conditional database sizes, and memory-access patterns. Theoretical analysis shows that VIFP achieves output-sensitive mining while avoiding the dominant cryptographic overhead of ORAM-based dynamic tree simulation. Future work will focus on implementing VIFP in a three-party SMPC backend, extending it to malicious security, optimizing packed FHE thresholding, and benchmarking against secure Apriori, encrypted support counting, and ORAM-based tree simulation.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[2] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[3] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[4] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, 2002.

[5] Murat Kantarcioglu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, 2004.

[6] Tamir Tassa. Secure mining of association rules in horizontally distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):970–983, 2014.

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

[8] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 71:57–81, 2014.

[9] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Proceedings of ASIACRYPT*, pages 409–437, 2017.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33:34–91, 2020.

[11] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[12] Oded Goldreich. *Foundations of Cryptography, Volume 2: Basic Applications*. Cambridge University Press, 2004.

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

[14] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 299–310, 2013.

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Proceedings of EUROCRYPT*, pages 337–367, 2015.

[16] Kazuki Tanemura and Masato Oguchi. Privacy-preserving frequent pattern mining using FP-growth and fully homomorphic encryption. In *Proceedings of International Workshop / Conference on Secure Data Mining*, 2019.

[17] Xiaofeng Liu, Rongxing Lu, Jin Li, and Benjamin C. M. Fung. Privacy-preserving outsourced frequent pattern mining using fully homomorphic encryption. In *Proceedings of Secure Data Management*, 2015.

[18] Hiroki Imabayashi, Kazuki Tanemura, and Masato Oguchi. Efficient privacy-preserving frequent pattern mining with ciphertext packing. In *Proceedings of International Conference on Cloud Computing and Big Data*, 2016.

[19] Jin Li, Xiaofeng Chen, and others. Cloud-aided privacy-preserving association rule mining. *IEEE Transactions on Cloud Computing*, 2010s.

[20] Ying Liu, Xiaofeng Chen, and others. Two-cloud privacy-preserving association rule mining over outsourced encrypted data. *IEEE / ACM Transactions*, 2023.

[21] Jia and coauthors. Multi-user privacy-preserving association rule mining with multi-key TFHE. *Journal / Conference Proceedings*, 2023.