RGeneSSE: Privacy Preserving Genomic Interval Queries via Range Searchable Symmetric Encryption on Dynamic Databases

Blinded For Review

***Abstract*—Advances in Next Generation Sequencing (NGS) technologies have made large-scale genomic data generation both rapid and cost-effective, fueling critical applications in personalized medicine, ancestry analysis, and Genome-Wide Association Studies (GWAS). However, outsourcing genomic data to untrusted cloud environments poses severe privacy risks, as existing protection mechanisms either compromise utility (e.g., through differential privacy) or impose prohibitive computational overhead (e.g., via homomorphic encryption or secure multi-party computation).**

In this work, we investigate the technical feasibility of privacy-preserving genomic interval queries and present a scalable encrypted range query framework for large genomic databases. Our approach builds on Searchable Symmetric Encryption (SSE) and introduces a new interpretation of encrypted range queries as disjunctions of canonical range covers, enabling efficient realization through a dynamic disjunctive SSE primitive.

We design `GeneSSE`, the first dynamic disjunctive SSE scheme offering linear storage overhead, constant-time updates, and sub-linear search complexity, while ensuring forward privacy and Type-I backward privacy. Building upon `GeneSSE`, we construct a multi-attribute range SSE scheme (called `RGeneSSE`) that mitigates leakage vectors exploited by recent range query attacks while also supporting dynamic encrypted databases. Our formal security analysis and prototype implementation demonstrate that efficient, leakage-resilient, and privacy-preserving genomic interval queries are practically realizable, establishing the feasibility of secure and scalable encrypted genomic data analytics

Furthermore beyond computational biology, our techniques enable efficient, privacy-preserving range-based and general Boolean queries over large-scale encrypted dynamic databases.

1. Introduction

Outsourcing Genomic Data and Privacy Threats. With the completion of the Human Genome Project and the advent of Next Generation Sequencing (NGS) technologies, sequencing the entire human genome has become both faster and more affordable, leading to an unprecedented surge in genomic data collection and analysis. This revolution has enabled transformative applications in personalized medicine, disease prediction, ancestry characterization, and genetic compatibility testing, motivating research institutions, healthcare providers, and commercial entities to increasingly outsource genomic data storage and computation to third-party cloud infrastructures for their scalability and cost efficiency [1]. However, this outsourcing paradigm introduces serious privacy and security risks, as genomic data is inherently identifiable and encodes sensitive information about an individual's disease susceptibility, ancestry, and phenotypic traits, as well as that of their biological relatives. Even anonymized genomic datasets are vulnerable to re-identification attacks, exemplified by the Golden State Killer case[1], where genetic identification was performed through a relative's data in a public genealogy database; or the hacking of personal data of more than $14,000$ users of the genetic testing company 23andMe, where the stolen information comprised individuals' names, years of birth, familial relationship tags, proportions of shared DNA with relatives, ancestry profiles, and their self-reported locations[2]. The privacy threat becomes even more acute when digital genomic data undergoes bioinformatics processing, including sequence alignment, variant querying, or large-scale genomic database searches on untrusted cloud platforms. For instance, consider a patient, Alice, who stores her sequenced genome on the cloud and consults a testing service run by Bob, who wishes to query her genome using publicly known disease markers to assess her risk for specific conditions. While Bob's computation is medically valid, Alice cannot trust his infrastructure not to misuse or leak her data, as sharing her raw genome would expose her complete biological blueprint, and even anonymization cannot prevent re-identification. Such scenarios underscore the urgent need for privacy-preserving cryptographic frameworks that allow secure computation, such as disease testing or genetic compatibility analysis over encrypted genomic data, ensuring that neither the service provider nor the underlying cloud gains access to sensitive genetic information while enabling scalable and trustworthy genomic analytics.

Privacy Preserving Genomic Data Processing. A variety of privacy-preserving techniques have been developed to secure genomic data processing, broadly categorized into cryptographic and statistical obfuscation-based approaches. Early cryptographic works, such as those by Atallah et al. [2], Jha et al. [3], and Troncoso-Pastoriza et al. [4],

1. https://www.latimes.com/california/story/2020-12-08/man-in-the-window

2. https://techcrunch.com/2023/12/04/23andme-confirms-hackers-stole-ancestry-data-on-

focused on privacy-preserving sequence comparison and pattern matching through secure two-party computation and garbled circuits [5] (GC). Subsequent research extended these ideas using homomorphic encryption (HE) and private set intersection (PSI) protocols [6], enabling privacy-preserving analyses for genetic compatibility testing, paternity verification, and disease-risk assessment. In parallel, differential privacy (DP) [7], [8], [9] has been employed to protect summary-level genomic statistics (e.g., minor allele frequencies, chi-square values) against membership inference attacks, though the added noise required for formal privacy guarantees often degrades data utility. More recent advances leverage powerful cryptographic frameworks such as secure multiparty computation (SMC) [10], [11], fully homomorphic encryption (FHE) [12], [13], [14], and hardware-assisted trusted execution environments (TEEs) [15], [16], [17], [18], [19] to provide privacy guarantees dependent on hardware assumptions.

Despite their promise, these methods face significant *limitations*: (1) SMC protocols introduce substantial communication/computation overhead, and require practical realization of the non-collusion assumption among its parties; (2) FHE-based schemes remain computationally expensive for large-scale genomic analyses, as the bootstrapping overhead scales with circuit size; (3) the injected noise in DP (needed for higher privacy budgets) often degrades data utility; (4) GC require re-running the expensive setup *every* circuit execution; and (5) TEEs are susceptible to side-channel and memory access-pattern leakage. Moreover, interoperability across heterogeneous cryptographic frameworks remains limited, as most solutions address specific genomic operations in isolation. This motivates the need for scalable, interoperable, and computation-efficient cryptographic frameworks that can support diverse genomic data analyses while preserving privacy end-to-end, thereby enabling secure, large-scale genomic research and applications without compromising individual confidentiality.

Motivation and Goals. In this work we focus on genomic interval queries: that are among the most common and powerful operations in bioinformatics, enabling analysts to retrieve and analyze genomic features (e.g., variants, genes, exons, regulatory elements) that overlap specific coordinate ranges. These queries are central to a wide spectrum of analyses, from identifying disease-associated mutations and annotating genomic variants to inspecting read coverage in NGS data and performing comparative genomics. However, performing such interval queries securely over outsourced or shared genomic databases presents serious privacy challenges. Traditional privacy-preserving methods, such as DP, while suitable for computing aggregated statistics, are unsuitable for precise range queries due to their accuracy loss through noise injection; whereas cryptographic methods like FHE, GC or SMC remain computationally or communication-wise prohibitive for large genomic datasets or require assumptions like non-collusion. Moreover, the access patterns in interval queries leak sensitive information about queried genomic regions or individual genotypes, negating the potential usage of TEEs.

Therefore, there is a pressing need for an encrypted genomic interval query framework that supports the following *goals*: (1) efficient, fine-grained range search directly on encrypted genomic data while preserving (2) query privacy, (3) access pattern confidentiality, and (4) scalability to large size genomic databases. Such a framework would enable secure retrieval of genomic records within specific coordinate ranges, such as *"fetch all variants in chr1:800000–801000"*, without revealing the query content or underlying genomic information, thereby bridging the gap between practical usability and strong cryptographic privacy guarantees in real-world genomic data analytics.

Our Contribution. The goal of this work is to evaluate the technical feasibility of implementing privacy-preserving genomic interval queries. With the established shortcomings of existing approaches (like FHE, GC, SMC, and TEEs), we investigate the applicability of a different cryptographic primitive to the problem of genomic interval queries. Towards this end we make the following contributions:

- **Cryptographic Design.** As a first step toward realizing privacy-preserving genomic interval queries, we translate this problem into a well-defined set of technical requirements and propose a novel cryptographic protocol that fulfills them. Conceptually, we interpret a genomic interval query (example: "*retrieve all variants within chr1:8000000-801000*") as a *range query* (i.e. *8000000-801000*) over a variable (i.e. *chr1*) in encrypted data. Thereafter, we connect the realization of such encrypted range queries to Searchable Symmetric Encryption (SSE), by representing encrypted range searches on genomic data as *disjunctions of canonical range covers* on carefully constructed range-tree data structures. Usage of SSE to realize genomic interval queries fulfills our intended *goals*: (1) careful problem definition allows genomic searches through SSE, thereby ensuring privacy; our SSE protocol ensures (2) query privacy and (3) access pattern confidentiality, while resisting state-of-the-art attacks and (4) being scalable to large databases. Beyond secure data retrieval, usage of SSE also allows seamless application to a range of downstream analyses, including disease-associated mutation detection, variant annotation, read coverage inspection in next-generation sequencing (NGS) data, and comparative genomics, since such applications are also interpretable as range searches. To the best of our knowledge, this is the first cryptographic framework to address privacy-preserving genomic interval queries through practical SSE.
- **Dynamic and Leakage-Resilient Range SSE.** We design a novel SSE protocol- `GeneSSE`- to act as the foundation of our privacy-preserving genomic interval queries. `GeneSSE` builds upon [20] but with crucial improvements: (1) we append the capability of operating over dynamic databases; and (2) we remove the requirement in [20] where *a-priori* knowledge of keyword frequencies is known. Both these improvements allow for practical databases where ge-

nomic information needs to be added as gene mapping extends to wider populations. GeneSSE achieves (i) linear storage overhead, (ii) constant-time updates, and (iii) sub-linear search complexity, while providing (iv) forward privacy and (v) Type-I backward privacy, properties that may be of independent interest. Thereafter, building upon GeneSSE as a black-box, we design a leakage resistant range SSE scheme (named RGeneSSE) supporting scalable multi-attribute range queries. RGeneSSE adopts a range-tree-based query decomposition approach similar to [21], but significantly strengthens privacy by mitigating critical leakage vectors specifically structure, volume, and search pattern leakages that prior designs are vulnerable to. *By translating genomic interval queries to range searches and transitively reducing them to disjunctions over canonical covering nodes* of a range-tree, our RGeneSSE achieves efficient, privacy-preserving range search on dynamic encrypted genetic databases while resisting state-of-the-art leakage-abuse attacks [21]. Further details are provided in Sections 2 and 3. We give a detailed analysis of the attacks in [21] and demonstrate that our proposed range SSE scheme is robust and resistant to these attacks.

- **Security Analysis and Implementation.** We present a detailed analysis of the leakage profile for GeneSSE and RGeneSSE, and prove their security formally. We practically validate the performance of GeneSSE by evaluating and comparing its search performance over the Enron email corpus with [20], [22], [23]. We also present a prototype implementation of RGeneSSE over genomic database maintained by the Genome in a Bottle Consortium of the National Institute of Standards and Technology, containing 1.930 GB of data in VCF 4.2 format (for the NA12878 genomic variant). The search overhead for finding ranges of magnitude over 10^4 is around 0.12 seconds. The experimental results are analyzed in detail in Section 4.

1.1. Mapping Genomic Interval Queries into Range Searches through SSE

Variant Call Format (VCF) Specification. In this work, we closely follow the Variant Call Format Specification (v4.2) for the specification of the genomic dataset. A sample is given in Figure 1. Each record has 8 fields: (1) CHROM that marks the chromosome number from the reference genome; (2) POS referring to the reference position *within* each CHROM; (3) unique identifier ID; (4) a reference base REF that can take one value out of A, C, G, T, N; (5) an alternate base ALT; (6) quality QUAL wrt. the assertion made in ALT; (7) whether the chromosome position has passed all filters; and (8) additional information.

A **trivial search** over such dataset could be realized as such: use a multi-attribute search protocol over the attributes CHROM and POS, to return all records within the designated chromosomes. This approach has a crucial drawback: For n chromosomes and m positions within each chromosome, such a trivial approach requires $O(n \cdot m)$ search queries[3].

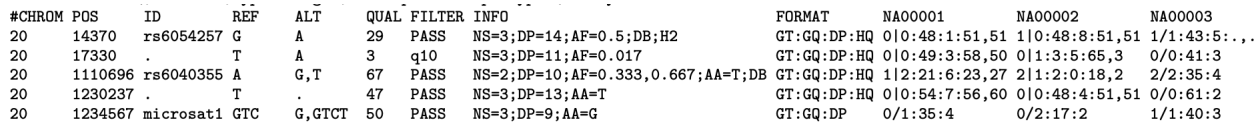
We however assert a further constraint from the dataset specification: **the positions POS within each chromosome CHROM are *sorted* numerically**. Thus instead of $O(n \cdot m)$ search queries using a multi-attribute search protocol, one could use $O(n)$ queries using a multi-attribute *range* search protocol and achieve the same result. Moreover, if the chromosomes of interest are also adjacent, a *single* (i.e. $O(1)$) range search *two* attributes, CHROM and POS- suffices. Thus, exploiting the structure of the VCF specification allows for mapping genomic interval queries to range searches, that leads to an asymptotic reduction in search queries executed.

Privacy-Preserving Range Searches. Next up is the question of performing such range searches on encrypted genomic datasets following the VCF specification; for that, we choose to rely on Searchable Symmetric Encryption (SSE). SSE [24], [25], [26], [27], [28] enables a resource-constrained client to perform keyword searches *directly* over symmetrically encrypted document collections stored on an *untrusted* server. While the strongest security guarantees are achievable via ORAM techniques [29], they incur substantial computational and communication overheads. Consequently, most practical SSE schemes trade off full obliviousness for efficiency by allowing limited leakage, such as database size, query, and access patterns, often mitigated using optimized ORAM variants [30], [31], [32], [33] or hardware-based approaches like Intel SGX [34], though the latter's security remains under active scrutiny [35], [36].

Prior research has extended SSE to support expressive queries such as conjunctions, disjunctions, Boolean, and range queries. Here, we focus exclusively on range queries because of the considered use-case. Faber et al. [37] proposed a range SSE scheme based on OXT [27], but it failed to account for false positives under data skewness—later addressed by Demertzis et al. [38], [39] through their Logarithmic SRC-i_1 and SRC-i_2 schemes. However, these constructions are limited to (1) single-attribute (1-D) queries and (2) static databases, *both unsuitable for genomic datasets investigated here*. Subsequent works [34], [40], [41], [42], [43], [44], [45], [46], [47], [48] also support only 1-D queries, while others [49], [50], [51], [52], [53] enable multi-attribute range queries on dynamic databases. Nevertheless, most existing range SSE schemes, static or dynamic, single or multi-attribute remain vulnerable to leakage abuse attacks, which are discussed next.

Attacks on Range SSE. Leakages in SSE schemes are typically classified as search, access, and volume patterns, which have been exploited in powerful database reconstruction attacks [54], [55], [56], [57]. For range queries, such leakages enable adversaries to infer plaintext order and values of encrypted records [46], [57], [58], [59], [60], [61], [62], [63], [64], [65], [66], [67], and are thereby disastrous

3. We do not assume the search query to have capabilities of performing *conjunctions*. Was that the case, the number of queries would reduce to $O(n)$, but communication complexity would still be $O(n \cdot m)$ because of the need to handle each conjunct separately.



#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	NA00001	NA00002	NA00003
20	14370	rs6054257	G	A	29	PASS	NS=3;DP=14;AF=0.5;DB;H2	GT:GQ:DP:HQ	0|0:48:1:51,51	1|0:48:8:51,51	1/1:43:5:.,.
20	17330	.	T	A	3	q10	NS=3;DP=11;AF=0.017	GT:GQ:DP:HQ	0|0:49:3:58,50	0|1:3:5:65,3	0/0:41:3
20	1110696	rs6040355	A	G,T	67	PASS	NS=2;DP=10;AF=0.333,0.667;AA=T;DB	GT:GQ:DP:HQ	1|2:21:6:23,27	2|1:2:0:18,2	2/2:35:4
20	1230237	.	T	.	47	PASS	NS=3;DP=13;AA=T	GT:GQ:DP:HQ	0|0:54:7:56,60	0|0:48:4:51,51	0/0:61:2
20	1234567	microsat1	GTC	G,GTCT	50	PASS	NS=3;DP=9;AA=G	GT:GQ:DP	0/1:35:4	0/2:17:2	1/1:40:3

Figure 1: Sample of VCF v4.5 (Source:https://www.nist.gov/programs-projects/genome-bottle)

for the use-cases of genomic data. Recent work by Falzon et al. [21] showed that existing multi-attribute range SSE schemes leak volume, search, and structure patterns, allowing near-complete reconstruction with minimal overhead. Defense strategies include hardware-based solutions using TEEs (e.g., Intel SGX) [68], [69], which remain vulnerable to side-channel attacks [70], [71], oblivious constructions like ORAM [29], [46], which are computationally expensive [72], [73], and protocol-level optimizations [37], [38], [39], [44], [45], [47], [48], [53], most of which remain vulnerable [21].

Despite extensive research, no existing work achieves a scalable, leakage-resilient, dynamic multi-dimensional range SSE resistant to recent inference attacks; thereby, to the best of our knowledge, existing range SSE schemes do not satisfy the expectations of privacy required for genomic applications. This motivates our central question:

Can we design a ① leakage-resilient, ② dynamic SSE scheme that supports ③ multi-dimensional range queries while remaining ④ efficient and ⑤ scalable to large genomic databases?

1.2. Technical Overview: RGeneSSE

In this work, we achieve all five objectives. We observe that existing range SSE schemes, which query the SSE index for each node in a range cover, are inherently leaky, necessitating a fundamental shift in perspective. To our knowledge, we are the first to employ a *disjunctive* SSE approach to realize range queries (objective ③ above). Specifically, we leverage TWINSSE [20], whose twin-index structure is well-suited to mitigating leakage in range searches (objective ① above). Since TWINSSE is originally designed for static databases, we introduce several non-trivial extensions to make it dynamic (objective ② above), without compromising its leakage-resilient structure. This enables the construction of a dynamic range SSE that supports multi-dimensional queries (objective ④ above) and is scalable to large genomic databases (objective ⑤ above). Our key contribution lies in this perspective shift, moving away from inherently leaky designs toward a robust and efficient dynamic range SSE framework for multi-attribute encrypted databases.

Range queries via Dynamic, Disjunctive SSE: A new perspective. Existing range SSE schemes relying on range-tree structures and range covers inherently suffer from *structure pattern leakages*, enabling inference of sub-query co-occurrences from individual search tokens.

We propose a paradigm shift in range SSE design. **Our key insight is that a range query can be *reduced to a disjunction* over the nodes of a range cover, enabling its execution via a *disjunctive SSE*.** Unlike prior approaches, this eliminates the need to issue separate searches for each node. In the dynamic database model, we show that such a dynamic disjunctive SSE scheme[4]- named GeneSSE- can securely execute such disjunctive queries over encrypted indexes to retrieve matching encrypted document identifiers. GeneSSE is then used to instantiate RGeneSSE, by reducing a range query to disjunctions over its range cover. This structural shift inherently mitigates structure pattern leakages and thwarts attacks like [21] by preventing leakage of individual token result volumes and prohibiting construction of a *volume map*, thus offering a "by-design" defense mechanism. However, there still remain roadblocks in concretely realizing range queries as disjunctions, which constitute the core problems we provably solve in this work.

Problem-I (Leakages): The main challenge is to hide the one-to-one mapping between covering nodes and search tokens in a disjunction. Although a generic disjunctive SSE prevents creation of a *volume map*, it still leaks the *overall* result volume and the memory access pattern of search tokens, enabling the server to infer token query frequencies across searches.

Solution. We instantiate GeneSSE in a similar manner as [20], thereby structurally mitigating overall volume pattern leakages [58], [62], [65] and outperforms state-of-the-art disjunctive SSE [22], [74] in storage while maintaining comparable search cost. GeneSSE supports general Boolean queries (CNF/DNF) using "super keywords" (specialized as "meta-keywords"), disjunctions of selected database keywords. A disjunctive query q is transformed into a conjunctive query q_{mkw} of meta-keywords, executed via a conjunctive SSE oracle, returning a *super set* of the true result. This transformation (i) obfuscates the mapping between covering nodes and search tokens, and (ii) suppresses overall volume and access pattern leakages through GeneSSE's design and proven security guarantees.

Problem-II (Functionality): While the construction in [20] supports disjunctions, it lacks support for dynamic updates. Hence, a black-box usage of [20] is impossible in the instantiation of GeneSSE.

Solution. One functional requirement for the construction in [20] is the a-priori knowledge of keyword frequencies: an assumption that is not practical for dynamic settings (like genetic databases, where functionality for modifications, e.g. addition of records, is paramount). We therefore propose a

4. Closest to our approach is Faber et al. [37], which models range queries as disjunctions over a conjunctive SSE, resulting in inefficiencies, leakage, limited multi-attribute and functionality only over static databases.

Notations	Meaning
λ	security parameter
n	maximum number of keywords in a disjunctive query
id	document identifier
w	a keyword
N	Total number of records in the database
$\mathcal{W}$	dictionary of keywords $\mathcal{W} = \{\mathsf{w}_1, \ldots, \mathsf{w}_N\}$
$\mathcal{D}$	domain space
d	total attributes in the database ($\lvert\mathcal{D}\rvert$)
attr	an attribute in $\mathcal{D}$
val	value of an attribute corresponding to a record id
$\mathtt{W}$	covering set of nodes
DB	database $(\mathsf{id}_j, (\mathsf{attr}_i, \mathsf{val})_{i=1,j=1}^{d,\quad N}) \in$ **DB**
$\lvert\mathbf{DB}(\mathsf{w})\rvert$	number of documents containing w
$\mathtt{T}$	range tree data structure that stores all attr-val pairs in **DB**
v	node in the tree $\mathtt{T}$
$\mathcal{R}_q$ or $\mathbf{DB}(q)$	result set returned to the client after a query q.
$x \xleftarrow{\$} \chi$	uniformly sampling x from χ
$x = \mathcal{A}$	x is output of a deterministic algorithm
$x \leftarrow \mathcal{A}'$	x is output of a randomized algorithm
$\mathbb{Z}_q^*$	multiplicative group modulo some prime q
F	Pseudo-random function with output range in $\{0,1\}^\lambda$
F_p	Pseudo-random function with output range in a multiplicative group modulo prime p

TABLE 1: Notations

novel dynamic disjunctive SSE- GeneSSE- that eliminates the need for apriori keyword frequency knowledge, while preserving optimal update and search complexity and linear storage overhead. To our knowledge, GeneSSE is the first dynamic disjunctive SSE achieving (1) linear storage, (2) constant update, (3) sub-linear search, (4) forward privacy, and (5) Type-I backward privacy. Further details appear in Section 2. RGeneSSE is a natural extension thereof.

Note. To test the attack resilience of RGeneSSE, we adopt the attack framework of [21]: the most recent and efficient attack on multi-dimensional range queries. While prior works [46], [57], [59], [60], [61], [62], [63], [64], [65], [66], [67] primarily target 1-D settings (with limited 2-D exploration [66]) by exploiting search and volume patterns, [21] introduces an additional *structure pattern* leakage enabling highly efficient multi-attribute attacks. We therefore focus on defending against [21] and demonstrate robustness of our construction against search, volume, and structure pattern leakages.

Notations and Other Prerequisites: We describe the notations used throughout the paper in Table 1. We also capture detailed backgrounds for Dynamic SSE syntax/correctness/security in Appendix A, and overview of [20] in Appendix B due to lack of space.

2. GeneSSE: A Dynamic Disjunctive SSE

We now present our dynamic disjunctive SSE (i.e. GeneSSE) construction, that achieves (1) linear storage, (2) constant updates, (3) sub-linear search, and (4) forward and (5) backward privacy, the first such dynamic disjunctive SSE construction to our knowledge.

2.1. Challenges and Solutions

In the previous section, we gave an informal notions of the challenges in designing GeneSSE. We formalize them now.

Challenge 1: Updating Multiple Meta-Keywords for Every Keyword Update. Since GeneSSE is instantiated in a manner similar to [20], it also has a notion of "meta-keywords". A meta-keyword mkw_i is a disjunction of carefully chosen keywords from **DB** ($\mathsf{mkw}_i = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$). For a database with N keywords, there exists an $O(N)$-sized set $\mathcal{S}$ of meta-keywords such that any disjunctive query $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$ can be represented as a conjunction of meta-keywords $q' = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$ with $\mathbf{DB}(q) \subseteq \mathbf{DB}(q')$ (see [20] for a formal proof). Such a "meta-keyword" construct suffices for static databases (like [20]), but immediately poses problems for dynamic databases (like GeneSSE). This is because each keyword w corresponds to $O(n'^2)$ meta-keywords[5], so a single keyword update would require $O(n'^2)$ meta-keyword updates, leading to quadratic overhead in update complexity, which is undesirable.

Solution. A naive approach of solving this problem could be to store $(\mathsf{mkw}, \mathsf{w})$ pairs; however, this would leak the correspondence between keywords and meta-keywords, voiding the privacy guarantees of GeneSSE. Instead, in GeneSSE, we parameterize the bucket size $n' = O(1)$, which is sufficient in practice (e.g., 5–10 [20]). This choice preserves correctness, ensures asymptotic update complexity comparable to state-of-the-art literature, and incurs only a constant-factor slowdown, as formally proved in Theorem 1 and empirically validated in Section 4.

Stronger Backward Privacy. By updating all $O(n'^2)$ meta-keywords per keyword update, correlations between updates and keywords are eliminated. Consequently, GeneSSE achieves Type-I Backward Privacy[6], a stronger guarantee over Type-II Backward Privacy[7], as formally established in Appendix F.

Challenge 2: Loss of Correctness from Multiple Keyword Updates in a Meta-Keyword. Even after addressing Challenge 1, each meta-keyword mkw is updated for every update to its constituent keywords. This leads to a loss in correctness when multiple keywords update the same mkw. Example: If one meta-keyword mkw_i contains two keywords w_1 and w_2, updates to w_1 risk updating the associations of mkw_i-w_2 for $\{\mathsf{mkw}_i\}$: being set of meta-keywords against keyword w_2 unless careful design is enforced.

Solution. To prevent overwrites while preserving GeneSSE's structural properties, we encode the w-mkw association in two components: (i) a blinding factor α stored in the inverted search index at the server, which uniquely identifies updates for each keyword; and (ii) a cross-term stored in the XSet, which is also computed obliviously by the server during conjunctive search operations. For each update, a unique $\mathtt{xtag}$ is generated that incorporates these components, *ensuring that updates from multiple keywords to the same meta-keyword do not interfere with each other*. In other words, for our example, updates to w_1 do not update the mkw_i-w_2 association. This design preserves all

5. Where n' is the bucket size of keywords based on frequency, as originally in [20].

6. No correlation between an update operation and its corresponding keyword can be inferred

7. Leakage of correlations between an update and its keyword

update information for all keywords, maintains the integrity of meta-keyword queries, and guarantees correctness of GeneSSE. Importantly, it avoids decomposing mkw into constituent keywords, which would otherwise leak keyword-to-meta-keyword associations and violate the privacy guarantees of GeneSSE(cf. Algorithm 3).

Challenge 3: Apriori Knowledge of Keyword Frequency. The original construction in [20] requires a-priori knowledge of keyword frequencies to construct the meta-keyword database, which is feasible only in the static database model. In a dynamic setting like genomic databases, genetic records (and thus keywords) are added or deleted over time, and their frequency depends on the sequence of updates, making black-box use of a static SSE infeasible.

Solution. In the static database construction [20], keyword frequencies are used for *ordered* bucketization to optimize search complexity by reducing the inclusion of high-frequency meta-keywords. This ordering, however, does not affect meta-keyword generation or query correctness. In GeneSSE, we adopt **unordered bucketization**, where keywords are adaptively assigned to buckets as they are added, while preserving correctness[8]. While unordered bucketization may include high-frequency meta-keywords in worst-case queries, empirical results in Section 4 show that average-case search complexity remains comparable to the static case with ordered bucketization [20]: being linear in the number of meta-keywords with the least updates in the query. This adaptation, though slightly increasing worst-case search cost, enables richer functionality by supporting fully dynamic databases.

Algorithm 1 GeneSSE.Setup

Input: λ
Output: sk, st, **EDB**
1: **function** GeneSSE.Setup(λ)
2: **Client**
3: Sample a uniformly random key K_T for PRF F
4: Sample uniformly random keys K_X, K_Y, K_Z for PRF F_p
5: Initialize UC, TSet, XSet to empty maps
6: Set sk $= (K_T, K_X, K_Y, K_Z)$, $st = (\mathtt{UC}, n')$
▷ n' is bucket size for unordered bucketization (Section 2.1)
7: Set **EDB**= (TSet,XSet)
8: Send **EDB** to the server. Client keeps {sk, st}.

2.2. Technical Framework

Having addressed the plausible challenges in designing GeneSSE, we formally define the construction in this section. The GeneSSE framework is instantiated along the lines of [20] (with changes as discussed in Section 2.1). It can be defined formally as a client-server protocol comprising of three algorithms, $\Pi_{\mathtt{GeneSSE}} = \{\mathtt{Setup}, \mathtt{Update}, \mathtt{Search}\}$. We define each algorithm below; statements marked in red are our changes to [20].

- GeneSSE.Setup(λ): As detailed in Algorithm 1, this deterministic client-side algorithm takes a security parameter λ and initializes the system by (1) sampling PRF keys sk for the encrypted search index and search token generation, (2) initializing empty multi-maps TSet and XSet, and (3) setting up client state st, which includes a counter UC to track updates to meta-keywords and a bucket size n' that partitions the database keywords.
- GeneSSE.Update(sk, st, op, (id, w)): This client-server protocol handles updates (op, (id, w)). The client computes meta-keywords for w using the modified Gen_mkw routine [20] (Algorithm 2, modifications marked in red), updating only the bucket containing w. The client then invokes the modified Update routine (Algorithm 3) to generate addresses, values, α, and cross-tags, which the server stores in TSet and XSet.
- GeneSSE.Search(q, sk, st, **EDB**): The client converts a disjunctive query q into a meta-query q_{mkw} using GenMQuery [20], sorting keywords based on update count rather than frequency (Challenge 3, Section 2.1). The client then executes the modified Search protocol (Algorithm 5, modifications in red) on the server to retrieve the encrypted document identifiers corresponding to q.

Note. In GeneSSE.Search (lines 15-17, Algorithm 5), **xtokens** are generated for all (w, mkw) pairs. Since each

Algorithm 2 GeneSSE.Gen_mkw

Input: w, st
Output: $\mathsf{mkw}_{\mathtt{list}}$
1: **function** GeneSSE.Gen_mkw(w, st)
2: Parse $\mathsf{st} = n'$
3: $\Delta \leftarrow \Delta \cup \{\mathsf{w}\}$
4: $n_{\mathsf{B}} = \lceil \frac{|\Delta|}{n'} \rceil$
5: Partition Δ as $\{\Delta_1, \cdots, \Delta_{n_{\mathsf{B}}}\}$ such that $\Delta_i = \{\mathsf{w}_{(i-1)n'+1}, \cdots, \mathsf{w}_{in'}\}$ ▷ *The last bin may not contain n' ws. Keep only as many are left.*
6: Find the bucket Δ_k such that $\mathsf{w} \in \Delta_k$
7: Set $\ell \leftarrow |\Delta_k|$
8: **for** $i \leftarrow 1$ to ℓ **do**
9: **for** $j \leftarrow 0$ to $\ell - i$ **do**
10: $\mathsf{mkw}^{(k)}_{i,i+j} \leftarrow \Delta_k \setminus \{w^k_i, \ldots, w^k_{i+j}\}$
11: $\mathsf{mkw}_{\mathtt{list}} \leftarrow \mathsf{mkw}_{\mathtt{list}} \cup \{\mathsf{mkw}^{(k)}_{i,i+j}\}$
12: **return** $\mathsf{mkw}_{\mathtt{list}}, n_{\mathsf{B}}, \Delta$

Algorithm 3 GeneSSE.Update

Input: sk, st, op, (id, w), **EDB**
Output: **EDB**
1: **function** GeneSSE.Update(sk, st, op, (id, w))
2: **Client**
3: Parse $\mathsf{sk} = (K_T, K_X, K_Y, K_Z)$ and $\mathsf{st} = (\mathtt{UC}, \mathsf{n}')$
4: $\mathsf{mkw}_{\mathtt{list}}, n_{\mathsf{B}}, \Delta = \mathtt{Gen\_mkw}(\mathsf{w}, \mathsf{st})$
5: $\mathsf{st} = \mathsf{st} \cup \{n_{\mathsf{B}}, \Delta\}$
6: **for** $\mathsf{mkw} \in \mathsf{mkw}_{\mathtt{list}}$ **do**
7: **if** UC[mkw] is NULL **then**
8: Set UC[mkw] $= 0$
9: UC[mkw] $=$ UC[mkw] $+ 1$
10: addr $= F(K_T, \mathsf{mkw}||\mathtt{UC}[\mathsf{mkw}]||0)$
11: val $= (\mathsf{id}||\mathsf{op}) \oplus F(K_T, \mathsf{mkw}||\mathtt{UC}[\mathsf{mkw}]||1)$
12: $\alpha = F_p(K_Y, \mathsf{id}||\mathsf{op}) \cdot (F_p(K_Z, \mathsf{mkw}||\mathsf{w}||\mathtt{UC}[\mathsf{mkw}]))^{-1}$
13: $\mathbf{xtag} = g^{F_p(K_X, \mathsf{mkw}||\mathsf{w}) \cdot F_p(K_Y, \mathsf{id}||\mathsf{op})}$
14: Send (addr, val, α, **xtag**) to the server
15: **Server**
16: Parse **EDB**= (TSet,XSet)
17: Set TSet[addr] $=$ (val, α)
18: Set XSet[**xtag**] $= 1$

8. Bucket size is set to 5 as per Challenge 1 solution

Algorithm 4 GeneSSE.GenMQuery

Input: $q = \{\mathsf{w}_{i_1} \vee \mathsf{w}_{i_2} \vee \ldots \vee \mathsf{w}_{i_l}\}$, st
Output: Meta-Query q_{mkw}

function GeneSSE.GenMQuery(q, st)
  Parse st $=(n', n_{\mathsf{B}}, \Delta)$
  Parse Δ as $\{\Delta_1, \ldots, \Delta_{n_{\mathsf{B}}}\}$
  Sort ws in q in increasing order of UC
  Partition query q into set of sub-queries P_q as $q_1 || \cdots || q_{n_{\mathsf{B}}}$, such that q_k contains ws only from Δ_k for $k = 1, \cdots, n_{\mathsf{B}}$
  for $q_k \in \mathsf{P}_q$ **do**
    Parse q_k as $\{w^k_{i_1}, \cdots, w^k_{i_{l'}}\}$
    for $j \leftarrow 1$ to l' **do**
      $\mathsf{mkw}^{(k)}_{i_{j-1}+1, i_j-1} \leftarrow \Delta_k \setminus \{w^k_{i_{j-1}+1}, \ldots, w^k_{i_j-1}\}$
      ▷ *Recall that* $\mathsf{mkw}_{i_k, j_k} \leftarrow \phi$ *if* $i_k > j_k$, $i_0 = 0$ *and* w^k_1 *is the first keyword in* Δ_k
    $\mathsf{mkw}^{(k)}_{i_{l'}+1, n'} \leftarrow \Delta_k \setminus \{w^k_{i_{l'}+1}, \ldots, w^k_{n'}\}$
    $q_{\mathsf{mkw},k} \leftarrow \mathsf{mkw}^{(k)}_{1, i_1-1} \wedge \ldots \wedge \mathsf{mkw}^{(k)}_{i_{l'}+1, n'}$
    $q_{\mathsf{mkw}} \leftarrow q_{\mathsf{mkw}} \vee q_{\mathsf{mkw},k}$
  return q_{mkw}

Algorithm 5 GeneSSE.Search

Input: q, sk, st, **EDB**
Output: Result set **DB**(q)

function GeneSSE.Search(q, sk, st, **EDB**)
  Client
  Generate $q_{\mathsf{mkw}} = (\vee_{k \in [n_{\mathsf{B}}]} q_{\mathsf{mkw},k}) = \mathsf{GenMQuery}(q, \mathsf{st})$
  Parse sk $= (\mathsf{K_T}, \mathsf{K_X})$ and st $= (\mathtt{UC}, n', n_{\mathsf{B}}, \Delta)$
  Use UC to identify meta-keyword with the least updates (assumed to be mkw^k_1 w.l.o.g)
  Initialize stokenList to an empty list
  Initialize $\mathtt{xtokenList}_1, \ldots, \mathtt{xtokenList}_{\mathtt{UC}[\mathsf{mkw}^k_1]}$ to empty lists
  for $j = 1$ to $\mathtt{UC}[\mathsf{mkw}_1]$ **do**
    $\mathtt{saddr}_j = F(K_T, \mathsf{mkw}_{1,1} || j || 0)$
    $\mathtt{stokenList} = \mathtt{stokenList} \cup \{\mathtt{saddr}_j\}$
    for $i = 2$ to n **do**
      for $\forall \mathsf{w}_{k',p} \in \mathsf{mkw}^{k'}_i$ and $\mathsf{w}_{k,l} \in \mathsf{mkw}^k_1$ **do**
        $\mathtt{xtoken}_{(i,p),l,j} = g^{F_p(K_X, \mathsf{mkw}^{k'}_i || \mathsf{w}_{k',p}) \cdot F_p(K_Z, \mathsf{mkw}^k_1 || \mathsf{w}_{k,l} || j)}$
          ▷ $k' \in [n_{\mathsf{B}}]; p \rightarrow \#\mathsf{w} \in \mathsf{mkw}^{k'}_i; l \rightarrow \#\mathsf{w} \in \mathsf{mkw}^k_1$
        $\mathtt{xtokenList}_{i,j} = \mathtt{xtokenList}_{i,j} \cup \{\mathtt{xtoken}_{(i,p),l,j}\}$
      for $\forall \mathsf{w}_{k',p} \notin \mathsf{mkw}^{k'}_i$ and $\mathsf{w}_{k,l} \notin \mathsf{mkw}^k_1$ **do**
        $\mathtt{xtoken}_{(i,p),l,j} = g^{\beta \cdot \gamma}$ ▷ $\beta, \gamma \xleftarrow{\$} \mathbb{Z}^*_p$
        $\mathtt{xtokenList}_{i,j} = \mathtt{xtokenList}_{i,j} \cup \{\mathtt{xtoken}_{(i,p),l,j}\}$
      $\mathtt{xtokenList}_j = \mathtt{xtokenList}_j \cup \{\mathtt{xtoken}_{i,j}\}$
    Randomly permute the tuple-entries of $\mathtt{xtokenList}_j$
  Send $(\mathtt{stokenList}, \mathtt{xtokenList}_1, \ldots, \mathtt{xtokenList}_{\mathtt{UC}[\mathsf{mkw}_1]})$
  Server
  Parse **EDB** $=$ (TSet, XSet)
  Initialize sEOpList to a empty list
  for $j = 1$ to stokenList.size **do**
    $\mathtt{cnt}_j = 1$
    $(\mathtt{sval}_j, \alpha_j) = \mathsf{TSet}[\mathtt{stokenList}[j]]$
    for $i = 2$ to n **do**
      $\mathtt{xtokenList}_{i,j} = \mathtt{xtokenList}_j[i]$
      for $k = 1$ to $|\mathtt{xtokenList}_{i,j}|$ **do**
        $\mathtt{xtag}_{(i,j),k} = (\mathtt{xtokenList}_{i,j}[k])^{\alpha_j}$
        if $\mathsf{XSet}[\mathtt{xtag}_{(i,j),k}] == 1$ **then**
          $\mathtt{cnt}_j = \mathtt{cnt}_j + 1$
    Set $\mathtt{sEOpList} = \mathtt{sEOpList} \cup \{j, \mathtt{sval}_j, \mathtt{cnt}_j\}$
  Send sEOpList to the client
  Client
  Initialize IDList and **DB**(q_{mkw}) to an empty list
  for $\ell = 1$ to sEOpList.size **do**
    $(j, \mathtt{sval}_j, \mathtt{cnt}_j) = \mathtt{sEOpList}[\ell]$
    Recover $(\mathsf{id}_j || \mathsf{op}_j) = \mathtt{sval}_j \oplus F(K_T, \mathsf{mkw}_1 || j || 1)$
    if op_j is add and $\mathtt{cnt}_j = n$ **then**
      Set $\mathtt{IDList} = \mathtt{IDList} \cup \{\mathsf{id}_j\}$
    else if op_j is del and $\mathtt{cnt}_j > 0$ **then**
      $\mathtt{IDList} = \mathtt{IDList} \setminus \{id_j\}$
  for each **DB**$(q_{\mathsf{mkw},k})$ from IDList **do**
    DB$(q_{\mathsf{mkw}}) =$ **DB**$(q_{\mathsf{mkw}}) \cup$ **DB**$(q_{\mathsf{mkw},k})$
  Client locally filters **DB**$(q) \subseteq$ **DB**(q_{mkw})

meta-keyword mkw consists of constituent keywords ws from a bucket, and **xtag**s are computed per $(\mathsf{w}, \mathsf{mkw})$ update, generating **xtoken**s only for present keywords ws would leak their association with mkw over multiple queries. To prevent this, we pad the **xtoken**s by including randomly sampled ws not in the mkw. With bucket size $O(1)$ (Section 2.1), this adds minimal overhead. The rest of the search proceeds with the client locally filtering the results to match the original disjunction.

Correctness of GeneSSE. The proof of correctness of GeneSSE follows from the correctness of the GeneSSE.GenMQuery function, noting that frequency based bucketization has no effect on the correctness of the scheme[9], and from the correctness of TSet instantiations from [27]. We state the following theorem for the correctness of GeneSSE.

Theorem 1 (Correctness of GeneSSE)**.** *Since the frequency-based bucketization in [20] has no effect on the correctness,* GenMQuery *correctly converts a given disjunctive query to a conjunction of appropriate meta-keywords in [20], and the* TSet *instantiation from [27] works correctly, hence* GeneSSE *satisfies correctness for an SSE scheme supporting disjunctive queries.*

Security Analysis of GeneSSE. The security of GeneSSE relys on secure PRFs and the decisional Diffie-Hellman assumption over $\mathcal{G}$. We formally prove simulation-based security against a well-defined leakage profile. Compared to prior dynamic SSE schemes, GeneSSE does not leak additional information; its meta-keyword structure obfuscates update patterns and provides stronger Type-I backward privacy. The main theorem is stated below:

Theorem 2 (Security of GeneSSE)**.** *Assuming that F and F_p are secure PRFs and the decisional Diffie-Hellman assumption holds over the group $\mathcal{G}$,* GeneSSE *is adaptively-secure with respect to a leakage function $\mathcal{L}_{\mathtt{GeneSSE}}$.*

Due to lack of space, we provide the formal proof of the correctness of GeneSSE in Appendix C, a rigorous leakage profile analysis (i.e. definition of $\mathcal{L}_{\mathtt{GeneSSE}}$), and a detailed simulation-based proof of security in Appendix D.1, D.2, and D.3 respectively.

Algorithm 6 RGeneSSE .Setup

Input: λ
Output: sk, st, **EDB**

function RGeneSSE .Setup(λ)
  Client
  sk, st, **EDB** $\leftarrow$ Dyn-TWINSSE.Setup(λ)
  Send **EDB** to the server. Client keeps {sk, st}.

3. RGeneSSE: Dynamic Volume Hiding Range SSE

We leverage the definitional index-based framework of a range SSE scheme to design our dynamic leak-

9. It is just an optimization choice made in [20]

age resilient range SSE scheme (RGeneSSE) for multi-dimensional ranges, that reduce a range query search to disjunctive keyword search. This framework allows us to leverage GeneSSE to support efficient multi-dimensional range searches in a dynamic setting. We explain the entire framework of RGeneSSE in the following steps -

Algorithm 7 RGeneSSE.InsertNode

Input: $\mathtt{T}, \mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}}$
Output: Updated tree T with the range inserted
1: **function** RGeneSSE.INSERTNODE($\mathtt{T}, \mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}}$)
2: **if** T == NULL **then**
3: $\mathtt{T}.\mathsf{val}_{\mathtt{low}} \leftarrow \mathsf{val}_{\mathtt{low}}$
4: $\mathtt{T}.\mathsf{val}_{\mathtt{high}} \leftarrow \mathsf{val}_{\mathtt{high}}$
5: **return** T
6: **if** $\mathsf{val}_{\mathtt{high}} < \mathtt{T}.\mathsf{val}_{\mathtt{low}}$ **then**
7: $\mathtt{T}.\mathsf{val}_{\mathtt{low}} \leftarrow \mathsf{val}_{\mathtt{low}}$
8: $\mathtt{T}.\mathtt{left} \leftarrow \mathtt{InsertNode}(\mathtt{T}.\mathtt{left}, \mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}})$
9: **else if** $\mathsf{val}_{\mathtt{low}} > \mathtt{T}.\mathsf{val}_{\mathtt{high}}$ **then**
10: $\mathtt{T}.\mathsf{val}_{\mathtt{high}} \leftarrow \mathsf{val}_{\mathtt{high}}$
11: $\mathtt{T}.\mathtt{right} \leftarrow \mathtt{InsertNode}(\mathtt{T}.\mathtt{right}, \mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}})$
12: **else**
13: $\mathtt{T}.\mathsf{val}_{\mathtt{low}} \leftarrow \min(\mathtt{T}.\mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{low}})$
14: $\mathtt{T}.\mathsf{val}_{\mathtt{high}} \leftarrow \max(\mathtt{T}.\mathsf{val}_{\mathtt{high}}, \mathsf{val}_{\mathtt{high}})$
15: $\mathtt{T}.\mathtt{subtree} \leftarrow \mathtt{InsertNode}(\mathtt{T}.\mathtt{subtree}, \mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}})$
16: **return** T

Range Supporting Data Structure. We adopt the range tree scheme from [21] to store attribute-value pairs of a d-dimensional database **DB**. A 1-D range tree is a binary search tree (BST), and a d-dimensional range tree is recursively constructed by associating each node in the BST of $[d_i]$ with the root of a BST on $[d_{i+1}]$, forming a hierarchical structure across dimensions. Nodes represent dyadic ranges. For a given range, we compute the minimal set of nodes using the *Best Range Cover* (BRC) technique, which, per THEOREM 5 of [21], guarantees a unique minimal cover with zero false positives.

The database **DB** consists of N records and d attributes, $\mathbf{DB} = \Sigma_{i=1,j=1}^{d,\ N}(\mathsf{id}_j, \{\mathsf{attr}_i, \mathsf{val}\})$. For each $(\mathsf{attr}\text{-}\mathsf{val})$ pair in domain d_i, the client generates a range tree and recursively builds subtrees for subsequent attributes. This structure is stored client-side to compute minimal covers for queries using $\mathtt{BRC}_{RT}$ [21], as detailed in Algorithms 9 and 10.

Encrypted Search Index. To reduce a range query to a disjunctive query and leverage disjunctive SSE lookups through GeneSSE, we construct an SSE-compatible inverted search index that is encrypted and stored at the server. For an update $(\mathsf{id}_j, \{\mathsf{attr}_i, \mathsf{val}\})$ of the i-th attribute and j-th record, the client computes the corresponding range tree node and adds id_j for $(\mathsf{attr}_i\text{-}\mathsf{val})$ to the search index (lines 8-9, Algorithm 9). For updates spanning multiple attributes, e.g., $(\mathsf{id}_j, \{\mathsf{attr}_i, \mathsf{val}\}, \{\mathsf{attr}_{i+1}, \mathsf{val}'\})$, the client generates the range tree for $(\mathsf{attr}_i, \mathsf{val})$ and its subtree for

Algorithm 8 RGeneSSE.BuildIndex

Input: $(\mathsf{id}_i, \mathsf{val})_{i=1}^{N}$
Output: DB
1: **function** RGeneSSE.BUILDINDEX($(\mathsf{id}_i, \mathsf{val})_{i=1}^{N}$)
2: **for** i = 1 to N **do**
3: **DB**[val] = id_i
4: **DB** = **DB** $\cup$ **DB**[val]
5: **return DB**

Algorithm 9 RGeneSSE.Update

Input: $\mathsf{sk}, \mathsf{st}, \mathtt{T}, \mathsf{op}, \mathsf{id}, \{\mathsf{attr}_i, \mathsf{val}\}_{i=1}^{d}$
Output: EDB
1: **function** RGeneSSE.Update($\mathsf{sk}, \mathsf{st}, \mathtt{T}, \mathsf{op}, \mathsf{id}, \{\mathsf{attr}_i, \mathsf{val}\}_{i=1}^{d}$)
2: **Client**
3: Initialize **DB** as an empty map
4: **for** $i = 1$ to d **do**
5: $\mathtt{T}_i.\mathtt{root} \leftarrow \mathtt{InsertNode}(\mathtt{T}_i.\mathtt{root}, \mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}})$
6: **if** $i < d$ **then**
7: $\mathtt{T}_i.\mathtt{root.subtree} \leftarrow \mathtt{InsertNode}(\mathtt{T}_i.\mathtt{root.subtree},$
8: $\mathsf{val}_{\mathtt{low}}, \mathsf{val}_{\mathtt{high}})$
9: **for** $i = 1$ to d **do**
10: **DB** $\leftarrow \mathtt{BuildIndex}(\mathsf{id}, \mathsf{val})$
11: **for** $i = 1$ to $d - 1$ **do**
12: **for** $v \in \mathtt{T}_i$ and $u \in \mathtt{T}_{i+1}$ **do**
13: **if** $v.\mathsf{val} \cap u.\mathsf{val} \neq \emptyset$ **then**
14: **for** $\mathsf{id} \in (v.\mathsf{val} \cap u.\mathsf{val})$ **do**
15: **DB** $\leftarrow \mathtt{BuildIndex}(\mathsf{id}, (v.\mathsf{val}||u.\mathsf{val}))$
16: **Client+Server**
17: **EDB** $\leftarrow \mathtt{GeneSSE.Update}(\mathsf{sk}, \mathsf{st}, n', \mathsf{op}, \mathsf{id}, \{\mathsf{w}_i\}_{i=1}^{d})$
18: **return EDB**

$(\mathsf{attr}_{i+1}, \mathsf{val}')$, storing id_j for both $(\mathsf{attr}_i\text{-}\mathsf{val})$ and $(\mathsf{attr}_i\text{-}\mathsf{val}) \cap (\mathsf{attr}_{i+1}\text{-}\mathsf{val}')$ (lines 10-14, Algorithm 9). This procedure is repeated for all subtrees of $(\mathsf{attr}_i\text{-}\mathsf{val})$. For a d-dimensional database, entries are stored for $d-1$ subtrees representing intersections $(d_i \cap d_{i+1})$, and the index is then encrypted and offloaded to the server.

Algorithm 10 RGeneSSE.Search

Input: $\mathsf{sk}, \mathsf{st}, \mathbf{EDB}, \mathtt{T}, v, q = [a_i, b_i]_{i=1}^{d}$
Output: $\mathbf{DB}(q)$
1: **function** RGeneSSE.SEARCH($q = [a_i, b_i]_{i=1}^{d}$)
2: **Client**
3: $\mathtt{W} \leftarrow \mathtt{BRC}_{RT}(\mathtt{T}, q, v)$ ▷ $\mathtt{BRC}_{RT}(\cdot) \rightarrow$ *[21]*
4: $q_{\mathtt{range}} = q_{\mathtt{range}} \cup \{\mathtt{W}\}$
5: **Client+Server**
6: $\mathbf{DB}(q_{\mathtt{range}}) \leftarrow \mathtt{GeneSSE.Search}(q_{\mathtt{range}}, \mathsf{sk}, \mathsf{st}, \mathbf{EDB})$
7: **Client**
8: Client locally filters $\mathbf{DB}(q) \subseteq \mathbf{DB}(q_{\mathtt{range}})$

Reduction From Range to Disjunction. For a d-dimensional range query, the client first computes the minimal cover in the first dimension (d_1) using BRC, then iteratively checks the subtrees in subsequent dimensions $(d_2, \ldots, d_d)$ for intersections with the query range. This process leverages $\mathtt{BRC}_{RT}$. The resulting set of covering nodes is converted into a disjunctive keyword query and sent to GeneSSE.Search (Algorithm 10). The returned result $(q_{\mathtt{range}})$ is a superset of the original query q, which the client filters locally. Correctness follows from the correctness of GeneSSE.Search.

Note. While [21] observes that the size of the BRC cover may vary in multi-dimensional queries, potentially leaking information about the range, RGeneSSE maps the covering nodes to keywords. The GeneSSE.Search algorithm computes meta-keywords for the disjunctive query and sends only these to the server, ensuring **no significant information about the queried range is revealed**.

3.1. Resilience against Reconstruction Attacks

In this section, we provide a detailed analysis of how RGeneSSE behaves under the BRC reconstruction attack of

range tree schemes as proposed in [21]. RGeneSSE is most relevant under this attack because it is structured as a range tree scheme that leverages the BRC range covering algorithm to find the minimal range cover corresponding to a multi-dimensional range query. However, we also emphasize the RGeneSSE is additionally resistant to the exploitation of structure pattern and search pattern leakages as extensively studied in [21].

Exploited Leakages. The attack in [21] targets leakages from the *tokenset* t, which the client generates to query the encrypted search index for each $w \in \mathcal{W}$, where $\mathcal{W}$ is the cover of a range query q obtained via BRC on the range tree. Key leakages include: (1) *volume pattern*, revealing the number of records returned per token $t \in$ t; (2) *search pattern* or *frequency map*, disclosing how often each token t appears across queries; and (3) *structure pattern*, capturing the tokenset and the response length for each token. The range tree reconstruction attack under BRC exploits both the volume and structure pattern leakages of the tokenset.

Reconstruction Attack Under BRC [21]. This attack exposes structural vulnerabilities in BRC-based range tree schemes, enabling information leakage even under structured encryption. It is a polynomial-time reconstruction attack that leverages query co-occurrence patterns to infer the underlying data structure. The attack starts by identifying inner nodes in the range tree using observed tokenset constraints. A co-occurrence graph is then built, where nodes represent tree elements and edges capture token relationships. Triangular structures in this graph allow differentiation between leaf, boundary, and inner nodes. Next, an inner grid reconstruction refines the tree layout and reorders records. Finally, volumes of extreme domain points are estimated by extrapolating missing values based on known token-volume relationships.

RGeneSSE **Leakage Profile.** Our multi-dimensional dynamic range SSE framework relies on two main data structures: (i) a range tree T that encodes the attribute-value pairs or range cover for the d-dimensional attribute space, and (ii) an inverted search index, which is a dictionary mapping meta-keywords (derived from specific attribute-value pairs in **DB**) to the corresponding record identifiers. Importantly, the inverted index is not a direct mapping from attribute-value pairs to records. By using GeneSSE.Update for constructing the encrypted search index, keywords in **DB** are transformed into meta-keywords, *obfuscating the one-to-one mapping with the tree nodes* (actual domain points). (1) *Query Processing:* For a given range query $q_{\texttt{range}}$, the client first computes the minimal range cover using the BRC algorithm. The nodes returned by this algorithm, representing minimal coverings of the actual domain data points, are treated as keywords to construct a disjunctive query $q_{\texttt{dis}}$. This query is passed to GeneSSE.Search, which transforms $q_{\texttt{dis}}$ into a conjunction of meta-keywords q_{mkw}. The server searches the encrypted database for q_{mkw} and returns the matching records. By the correctness of GeneSSE.Search, $\mathbf{DB}(q_{\texttt{dis}}) \subseteq \mathbf{DB}(q_{\mathsf{mkw}})$, ensuring the result covers the original query.

(2) *Leakage Analysis:* Reconstruction attacks [21] exploit volume pattern, search pattern, and structure pattern leakages. We discuss each in the context of RGeneSSE:

(a) *Volume Pattern:* Traditional attacks rely on the volume of records returned per token in the token set corresponding to each covering node. In RGeneSSE, the tokenset is a conjunction of meta-keywords used in a disjunctive SSE query. Consequently, the server only observes the overall result set for the query and **gains no information about the individual meta-keywords' partial results**[10]. Notably, (i) the individual tokens do not correspond to exact nodes returned by BRC, so they leak no information about precise domain points, and (ii) while the total query volume is visible, GeneSSE pads the result to an acceptable threshold (see Section 4), and the client locally filters the relevant documents. This design prevents the server from reconstructing the volume maps exploited in [21].

(b) *Search Pattern:* Since the tokens are meta-keywords without direct correspondence to exact domain points or attribute-value pairs, the frequency of tokens over multiple queries does not reveal meaningful information to the server and does not aid reconstruction.

(c) *Structure Pattern:* In prior attacks, structure pattern leakage arises from observing the tokenset and the sizes of partial responses per token. In RGeneSSE, the tokenset reveals no significant information about original domain points or the actual query range. Additionally, by reducing a range query to a disjunction and using a disjunctive SSE, the server only sees the complete result for the disjunction, eliminating sub-query or individual meta-keyword volume leaks.

Inapplicability of the Reconstruction Attack. From the above discussion, it is evident that the primary leakages required for the reconstruction attack to work are rendered inconsequential due to the (i) reduction of a range search to a disjunctive search; (ii) usage of GeneSSE that operates on a set of meta-keywords, thereby obfuscating the exact mapping of the tokens with the original data points in the domain. The Reconstruction Attack Under BRC [21] works in the following four steps. We explain each step and how they are inapplicable w.r.t. RGeneSSE-

(1) *Infer inner nodes*: This step exploits the individual tokens from a token set and infers information about the inner nodes of the range tree. **This non-trivial information is not possible to infer in** RGeneSSE**because the individual tokens of a tokenset is not the actual nodes in the range tree**, therefore does not provide any significant information.
(2) *Trim the co-occurrence graph*: This step exploits the number of times a pair of tokens appear in a tokenset. **This information is essentially stopped in** RGeneSSE**because of the reduction from range to disjunctive searches**.
(3) *Inner Grid Reconstruction*: This uses the co-occurrence graph and the information of the inner nodes to distinguish between the leaf and non-leaf nodes. **Since the co-occurrence graph itself cannot be formed as explained**

10. This is guaranteed by the security of GeneSSE

in the above step, inner grid reconstruction is also not inherently possible.
(4) *Inferring Extreme Node's Volume*: This step exploits the volume pattern leakages of individual tokens in a tokenset. Since this leakage is essentially stopped and rendered insignificant w.r.t. RGeneSSE, such inferences are not possible to make from our range SSE construction.

Correctness of RGeneSSE. We state the following theorem:

Theorem 3 (Correctness of RGeneSSE)**.** *Since* GeneSSE *is a functionally correct dynamic disjunctive SSE scheme, multi-dimensional range tree from [21] is correct, and Best Range Cover technique (BRC) from [21], [38], [53] is correct,* RGeneSSE*is a functionally correct range SSE scheme in the dynamic database model.*

The proof follows naturally from proof of correctness of GeneSSE (refer Section 2), and from corresponding proofs of correct operations of (1) multi-dimensional range tree in [21], and (2) Best Range Cover technique (BRC) in [21], [38], [53].

Security Analysis of RGeneSSE. The security analysis of RGeneSSE follows from the security guarantees of GeneSSE (that relies on security of secure PRFs and the decisional Diffie-Hellman assumption over the group $\mathcal{G}$). We formally state the following theorem, and provide a detailed simulation-based proof of security in Appendix G (due to space constraints).

Theorem 4 (Security of RGeneSSE)**.** *Since* RGeneSSE *is an adatively secure forward and backward private disjunctive SSE,* RGeneSSE *is adaptively-secure with respect to a leakage function* $\mathcal{L}_{\texttt{RGeneSSE}}$.

4. Experimental Results

We now describe the experimental setup used to evaluate the performance and analyze the storage and search complexity of GeneSSE and RGeneSSE as well as the low-level details of our prototype implementations.

Platform. The entire implementation of GeneSSE and RGeneSSE were done using C++ (with GCC 9 compiler), with Redis as the database backend. We ran the experiments on a *single* node with Intel Xeon E5-2690 v4 2.6 GHz CPU with 128 GB RAM and 512 GB SSD.

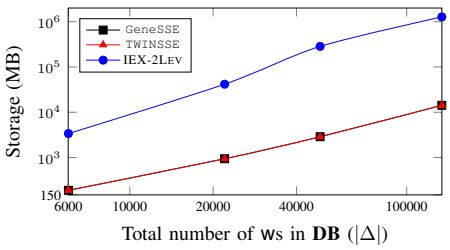


Figure 2: Server Storage (both axes are in log scale)

4.1. Performance Analysis of GeneSSE

In this section, we present the results of our experimental evaluations of GeneSSE and give comparison analysis with the performances of TWINSSE [20], ODXT [23] and IEX-2Lev [22], a state of the art disjunctive SSE in a static setting.

Dataset. We have used the Enron email dataset[11] for our experiments; it contains around 10K documents (emails) and 2.5 million thousand keyword-document pairs, with a total size of 1.3 GB.[12] We parse the raw Enron dataset and create a plaintext inverted index **DB** (of size approximately 650KB, which is 0.05% of the actual plaintext Enron database). The encrypted search index **EDB** is created as usual, post which GeneSSE is evaluated.

Storage Overhead Analysis. The storage overhead of GeneSSE is similar to the storage overhead of TWINSSE. Since GeneSSE inherits the fundamental structure of meta-keywords, it preserves the linear storage complexity of TWINSSE that marked a significant improvement form the quadratic storage overhead incurred by IEX-2Lev [22]. Figure 2 shows the linear increase in the storage of GeneSSE and TWINSSE compared with the quadratic increase in IEX-2Lev.

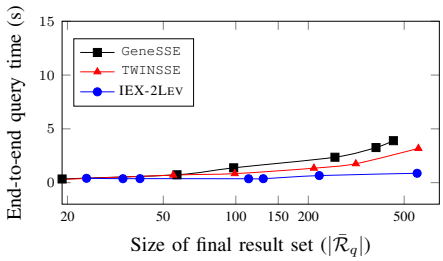


Figure 3: Comparison of end-to-end search latency vs final result size for disjunctive queries of the form $q = w_1 \vee w_2$ over Enron.

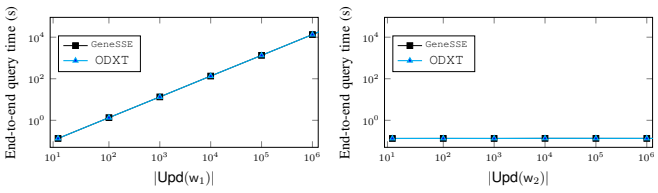
Disjunctive Search-time Overhead. We provide an end-to-end query performance comparison of GeneSSE with TWINSSE in Figure 3. For queries with smaller result sizes, GeneSSE achieves almost identical end-to-end query latency as TWINSSE. For queries with larger result sizes, TWINSSE performs slightly better. The slight, yet acceptable decrease in the computation time of GeneSSE is attributed to omitting the frequency-based bucketization strategy used in TWINSSE (refer Challenge 3 in Section 2.1). Also, GeneSSE maintains the linear increase in search complexity with increase in frequency (update count) of a keywords as shown by TWINSSE. We view this as an efficiency trade-off; we provide a forward and Type-I backward private dynamic disjunctive SSE, with the same linear

11. https://www.cs.cmu.edu/~enron/, https://www.kaggle.com/wcukierski/enron-email-dataset
12. The raw Enron dataset consists of folders of emails of individual employees. Since the emails are in plain English text, we performed a dictionary-based pre-processing to filter the keywords (we removed the stop-words and punctuation in the process), and created a list of (lexicographically ordered) keywords.

TABLE 2: Asymptotic Complexity Analysis and Comparison of RGeneSSE vs State-of-the-Art Range SSE. In the table, $\mathcal{W}$: set of all keywords in database **DB**; $\mathcal{W}_x$: largest sequence number of existing keyword ([53]), $|\ \mathtt{RC}\ |$: Number of keywords in the range cover/query q; $\mathtt{UC}(q)$: update count of all keywords in q; $\mathbf{DB}(q_{\mathsf{mkw}})$: result set of q_{mkw} i.e., transformed query from the disjunction of the range cover of q; **AR**: Attack Resistance against [21].

Scheme	Model	Update		Search		Communication		AR
		Client	Server	Client	Server	Update	Search	
[38]	Static	N/A	N/A	$2\mathcal{W}$	$O(\mathbf{DB}(q))$	N/A	$O(\mathbf{DB}(q))$	No
[44]	Dynamic	$O(\mathcal{W}_x)$	$O(\log\mathcal{W})$	$O(\mathcal{W}_x + \lvert\ \mathtt{RC}\ \rvert)$	$O(\lvert\ \mathtt{RC}\ \rvert)$	$O(\log\mathcal{W})$	$O(\lvert\ \mathtt{RC}\ \rvert)$	No
[53]	Dynamic	$O(\mathcal{W} + \log\ \lvert\ \mathcal{W}\ \rvert)$	$(\log\lvert\mathcal{W}\rvert)$	$O(\lvert\ \mathtt{RC}\ \rvert)$	$O(\mathtt{UC}(q))$	$O(\lvert\ \mathcal{W}\ \rvert)$	$\lvert\ \mathbf{DB}(q)\ \rvert$	No
This Work	Dynamic	$O(1)$	$O(1)$	$O\ \lvert\ \mathtt{RC}\ \rvert)$	$O(\lvert\ \mathbf{DB}(q_{\mathsf{mkw}})\ \rvert)$	$O(1)$	$O(\lvert\ \mathbf{DB}(q)\ \rvert)$	Yes

storage overhead as TWINSSE and maintains a linear search complexity. This is also a significant improvement when compared to other state-of-the-art disjuntive schemes like IEX-2Lev.



(a) (b)

Figure 4: (4a) End-to-End conjunctive query complexity for $q = \mathsf{w}_1 \wedge \mathsf{w}_2$, where the update count of w_1 is varied; (4b) End-to-End conjunctive query complexity for $q = \mathsf{w}_1 \wedge \mathsf{w}_2$, where the update count of w_2 is varied.

Conjunctive Search-time Overhead. Figure 4a compares the conjunctive search time of GeneSSE with ODXT: a linear increase in the search time complexity with an increase in the update count of w_1. This trend is similar to the search complexity of ODXT that scales with the increase in the update count of keyword in the query with the least updates (here w_1). This claim is further validated by Figure 4b: on varying $|\mathtt{UC}(\mathsf{w}_2)|$ the search complexity remains constant. This behaviour is reflected in GeneSSE as well.

Precision Analysis. In information retrieval, *precision* (denoted as η) refers to the proportion of relevant documents among the retrieved results. Figure 5 presents a comparison of the average precision values of $\bar{\mathcal{R}}_q$ for disjunctive queries (q) with varying keyword counts. Notably, the average precision remains above 85% in most cases, indicating that at least 85% **of the retrieved documents** in $\bar{\mathcal{R}}_q$ are relevant to the query q (i.e., they belong to the actual result set $\mathcal{R}_q$ without extraneous id-s). This level of precision closely matches that achieved by TWINSSE and is considered acceptable.

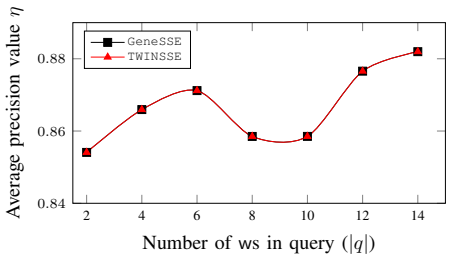


Figure 5: Average precision value vs number of keywords in a disjunctive query

Update Complexity. Recall from Challenge 1 in Section 2.1 that we set a $n' = O(1)$ bucket size; therefore, we have $O(n'^2)$ meta-keywords in each bucket. In our experimental evaluations, we consider $n' = 5$, this gives an optimal efficiency guarantee in terms of storage, search performance, precision and update complexity. Concretely, for a bucket size 5, any keyword update of the form $(\mathsf{op}, (\mathsf{id}, \mathsf{w}))$ corresponds to roughly 8 mkw_i that contains w. Thus for one update of w we make 8 meta-keyword updates. This remains uniform for all updates pertaining to any keyword in the database and also with different sizes of the database. Since for every atomic update of $(\mathsf{id}, \mathsf{w})$ a constant number of meta-keywords are updated always, the asymptotic update complexity of GeneSSE is $O(1)$ which is similar to the asymptotic update complexity of ODXT.

4.2. Performance Analysis of RGeneSSE

In this section, we present the results of our experimental evaluations of RGeneSSE and validate the asymptotic bounds on the performance overheads of RGeneSSE as shown in Table 2. Table 2 provides an asymptotic comparison of RGeneSSE with state-of-the-art range query schemes in terms of (1) database model, (2) update complexity (iff model is "Dynamic"), (3) search complexity (w.r.t. size of the range cover, or $|\ \mathtt{RC}\ |$), and (4) resistance to state-of-the-art attacks like [21]. We observe that RGeneSSE improves upon the update complexities of [53] and [44].

Dataset. We have used a genomic variant dataset maintained by the Genome in a Bottle Consortium of the National Institute of Standards and Technology, containing 1.930 GB of data in VCF 4.2 format (for the NA12878 genomic variant) with $3,893,572$ records. The data comprises of meta-information lines, a header line, and data lines each containing information about a position in the genome. The format also has the ability to contain genotype information on samples for each position. An example sample of the VCF format used is shown in Figure 6. We parse the raw VCF dataset and create a BRC tree and eventually map the internal nodes (keywords) and leaf nodes (document-identifiers) into a plaintext inverted index **DB** (of size approximately 71MB, which is 0.04% of the actual plaintext Enron database). The encrypted search index **EDB** is created as usual, post which RGeneSSE is evaluated.

Storage Overhead. The storage requirement of RGeneSSE asymptotically scales as $O(N + \log^d\ |\ \mathcal{D}\ |)$

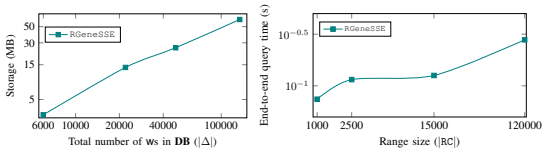
1.1 An example

```
##fileformat=VCFv4.2
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62436964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF    ALT     QUAL FILTER INFO                              FORMAT      NA00001        NA00002        NA00003
20     14370   rs6054257 G      A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2           GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20     17330   .         T      A       3    q10    NS=3;DP=11;AF=0.017               GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:3:5:65,3   0/0:41:3
20     1110696 rs6040355 A      G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2   2/2:35:4
20     1230237 .         T      .       47   PASS   NS=3;DP=13;AA=T                   GT:GQ:DP:HQ 0|0:54:7:56,60 0|0:48:4:51,51 0/0:61:2
20     1234567 microsat1 GTC    G,GTCT  50   PASS   NS=3;DP=9;AA=G                    GT:GQ:DP    0/1:35:4       0/2:17:2       1/1:40:3
```

This example shows (in order): a good simple SNP, a possible SNP that has been filtered out because its quality is below 10, a site at which two alternate alleles are called, with one of them (T) being ancestral (possibly a reference sequencing error), a site that is called monomorphic reference (i.e. with no alternate alleles), and a microsatellite with two alternative alleles, one a deletion of 2 bases (TC), and the other an insertion of one base (T). Genotype data are given for three samples, two of which are phased and the third unphased, with per sample genotype quality, depth and haplotype qualities (the latter only for the phased samples) given as well as the genotypes. The microsatellite calls are unphased.

Figure 6: Sample of VCF v4.2 used in our experiments

at the server to store all the attribute-value pairs with their corresponding records along with the records corresponding to each subtree for d dimensions in encrypted search index. We generate an inverted index of meta-keywords and their corresponding records for every subtree in each of d-th dimension. In Figure 7a we see that the storage overhead of RGeneSSE scales linearly with an increase in the number of keywords (meta-keywords) in the encrypted database.



(a) (b)

Figure 7: (7a) Server Storage (both axes are in log scale); (7b) Comparison of end-to-end search latency vs range size for range queries over GA4GH VCF file.

Search Overhead. RGeneSSE inherits the following properties from GeneSSE as it invokes GeneSSE.Update and GeneSSE.Search as black-boxes; (1) server-side updates are $O(1)$; (2) search time complexity for RGeneSSE increases linearly with an increase in the query range size ($|\mathtt{RC}|$). We implement the RGeneSSE algorithm, where the client issues a range of the form $[a, b]$, and generates a disjunction of covering nodes ($q_{\mathtt{range}}$) by invoking the $\mathtt{BRF}_{RT}$ algorithm. GeneSSE.Search is invoked subsequently with the generated disjunctive query $q_{\mathtt{range}}$. The server returns the set of encrypted records corresponding to ($q_{\mathtt{range}}$). In terms of search performance, RGeneSSE scales linearly with the search performance of GeneSSE, that in turn increases linearly with result size of the given meta-query q_{mkw}. This is validated in Figure 7b, where we plot the end-to-end search complexity of RGeneSSE with an increase in the size of the rage cover $|\mathtt{RC}|$. Since an increased range cover implies increase in the resultant records returned by the query, the linear increase in the search time with increase in $|\mathtt{RC}|$ implies a linear dependence of the search overhead of RGeneSSE on $|\mathbf{DB}(q_{\mathsf{mkw}})|$.

5. Conclusion

The proliferation of genomic data has revolutionized biomedical research but simultaneously magnified the risks of privacy breaches in outsourced genomic computation. Existing privacy-preserving approaches ranging from differential privacy to homomorphic encryption and trusted execution environments, either compromise analytical accuracy, incur prohibitive computational overhead, or remain vulnerable to access-pattern leakage.

In this work, we present RGeneSSE, the first scalable encrypted frameworks for privacy-preserving genomic interval queries. Our approach reinterprets range queries as disjunctions of canonical range covers, enabling efficient realization using dynamic disjunctive searchable symmetric encryption leveraging the GeneSSE construction. The resulting framework achieves linear storage, constant-time updates, sub-linear search, and strong privacy guarantees including forward and Type-I backward privacy, while mitigating leakage vectors exploited by recent range reconstruction attacks.

Our security analysis and prototype implementation confirm that fine-grained, leakage-resilient, and dynamic encrypted genomic query processing is not only theoretically sound but also practically viable. Beyond computational biology, our techniques generalize to privacy-preserving range-based and Boolean search over large-scale encrypted databases, offering a promising foundation for secure, scalable, and privacy-compliant data analytics in modern cloud-based computing environments.z

6. Ethical Consideration

We believe that it is justifiable to use the Enron dataset for our experiments given: (i) the extensive usage of this dataset for experiments in prior SSE literature [54], [57], [59], (ii) the fact that the data has long already been public, and (iii) there has been an effort by the researchers curating the version of the dataset used in this paper to remove users upon request.[13]

Our work involves processing and analysis of a genomic variant dataset maintained by the Genome in a Bottle Consortium of the National Institute of Standards and Technology. We have taken multiple measures to ensure ethical compliance and privacy protection throughout the study. (i) We use genomic datasets obtained from public reference standards (https://www.nist.gov/programs-projects/genome-bottle). No private or identifiable individual data was collected, shared, or re-identified by the authors. (ii) Our design aim to improve the privacy, security, and verifiability of genomic data processing, not to compromise it, and (iii) all experiments were performed in a controlled environment on local servers owned by the authors. We have carefully avoided publishing or releasing any information that could enable misuse, re-identification, or unauthorized inference attacks on existing genomic databases.

13. See https://www.cs.cmu.edu/~enron/ for detailed documentation.

References

[1] S. D. Goenka, J. E. Gorzynski, K. Shafin, D. G. Fisk, T. Pesout, T. D. Jensen, J. Monlong, P. C. Chang, G. Baid, J. A. Bernstein *et al.*, "Accelerated identification of disease-causing variants with ultra-rapid nanopore genome sequencing," *Nature Biotechnology*, 2022.

[2] M. J. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons." Association for Computing Machinery, 2003.

[3] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation." IEEE Computer Society, 2008.

[4] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient dna searching through oblivious automata." Association for Computing Machinery, 2007.

[5] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano, "Deriving genomic diagnoses without revealing patient genomes," *Science*, 2017.

[6] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, "Countering gattaca: efficient and secure testing of fully-sequenced human genomes," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*. Association for Computing Machinery, 2011.

[7] M. Hardt and K. Talwar, "On the geometry of differential privacy," in *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*. Association for Computing Machinery, 2010.

[8] S. E. Fienberg, A. Slavkovic, and C. Uhler, "Privacy preserving gwas data sharing." IEEE Computer Society, 2011.

[9] A. Johnson and V. Shmatikov, "Privacy-preserving data exploration in genome-wide association studies," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2013.

[10] O. Tkachenko, C. Weinert, T. Schneider, and K. Hamacher, "Large-scale privacy-preserving statistical computations for distributed genome-wide association studies," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018*.

[11] S. Carpov, K. Deforth, N. Gama, M. Georgieva, D. Jetchev, J. Katz, I. Leontiadis, M. Mohammadi, A. Sae-Tang, and M. Vuille, "Manticore: Efficient framework for scalable secure multiparty computation protocols," Cryptology ePrint Archive, Paper 2021/200, 2021.

[12] J. Zhang, Z. L.Jiang, P. Li, and S. M. Yiu, "Privacy-preserving multikey computing framework for encrypted data in the cloud," *Information Sciences*, 2021.

[13] S. Carpov, N. Gama, M. Georgieva, and D. Jetchev, "Genoppml - a framework for genomic privacy-preserving machine learning," in *IEEE 15th International Conference on Cloud Computing, CLOUD, 2022*.

[14] M. Namazi, M. Farahpoor, E. Ayday, and F. Pérez-González, "Privacy-preserving framework for genomic computations via multi-key homomorphic encryption," *Bioinformatics*, 2025.

[15] F. Chen, M. Dow, S. Ding, Y. Lu, X. Jiang, H. Tang, and S. Wang, "PREMIX: privacy-preserving estimation of individual admixture," in *AMIA 2016, American Medical Informatics Association Annual Symposium 2016*.

[16] F. Chen, C. Wang, W. Dai, X. Jiang, N. Mohammed, M. M. A. Aziz, M. N. Sadat, C. Sahinalp, K. E. Lauter, and S. Wang, "Presage: Privacy-preserving genetic testing via software guard extension," *BMC Medical Genomics*, 2017.

[17] F. Chen, S. Wang, X. Jiang, S. Ding, Y. Lu, J. Kim, S. C. Sahinalp, C. Shimizu, J. C. Burns, V. J. Wright, E. Png, M. L. Hibberd, D. D. Lloyd, H. Yang, A. Telenti, C. S. Bloss, D. Fox, K. Lauter, and L. Ohno-Machado, "Princess: Privacy-protecting rare disease international network collaboration via encryption through software guard extensions," *Bioinformatics*, 2016.

[18] S. Carpov and T. Tortech, "Secure top most significant genome variants search: iDASH 2017 competition," Cryptology ePrint Archive, Paper 2018/314, 2018.

[19] A. Mandal, J. C. Mitchell, H. Montgomery, and A. Roy, "Data oblivious genome variants search on intel SGX," *IACR Cryptol. ePrint Arch.*, 2018.

[20] A. Bag, D. Talapatra, A. Rastogi, S. Patranabis, and D. Mukhopadhyay, "Two-in-one-sse: Fast, scalable and storage-efficient searchable symmetric encryption for conjunctive and disjunctive boolean queries," *Proc. Priv. Enhancing Technol.*, 2023.

[21] F. Falzon, E. A. Markatou, Z. Espiritu, and R. Tamassia, "Attacks on encrypted range search schemes in multiple dimensions," *IACR Cryptol. ePrint Arch.*, 2022.

[22] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *EUROCRYPT 2017*.

[23] S. Patranabis and D. Mukhopadhyay, "Forward and backward private conjunctive searchable symmetric encryption," in *NDSS 2021*.

[24] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE S&P 2000*, 2000, pp. 44–55.

[25] E. Goh, "Secure indexes," *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[26] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT 2010*.

[27] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *CRYPTO 2013*, 2013, pp. 353–373.

[28] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *NDSS 2014*, 2014.

[29] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[30] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *IEEE S&P 2018*.

[31] E. Stefanov, M. van Dijk, E. Shi, T. H. Chan, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," *J. ACM*, 2018.

[32] D. Micciancio, "Oblivious data structures: Applications to cryptography," in *ACM STOC 1997*.

[33] X. S. Wang, K. Nayak, C. Liu, T. H. Chan, E. Shi, E. Stefanov, and Y. Huang, "Oblivious data structures," in *ACM CCS 2014*.

[34] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *HASP 2013*.

[35] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium, USENIX Security 2018*.

[36] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*.

[37] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *ESORICS 2015*, 2015, pp. 123–145.

[38] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. N. Garofalakis, "Practical private range search revisited," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference*, F. Özcan, G. Koutrika, and S. Madden, Eds., 2016.

[39] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, M. N. Garofalakis, and C. Papamanthou, "Practical private range search in depth," *ACM Trans. Database Syst.*, 2018.

[40] D. Bogatov, G. Kollios, and L. Reyzin, "A comparative evaluation of order-revealing encryption schemes and secure range-query protocols," 2019.

[41] F. Hahn and F. Kerschbaum, "Poly-logarithmic range queries on encrypted data with small leakage," in *Proceedings of the 2016 ACM on Cloud Computing Security Workshop, CCSW*, 2016.

[42] S. Kamara and T. Moataz, "SQL on structurally-encrypted databases," in *Advances in Cryptology - ASIACRYPT 2018 - 24th International*.

[43] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption with forward and stronger backward privacy." Springer-Verlag, 2019.

[44] C. Zuo, S. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security," in *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018*.

[45] F. Liu, K. Xue, J. Yang, J. Zhang, Z. Huang, J. Li, and D. S. Wei, "Volume-hiding range searchable symmetric encryption for large-scale datasets," 2024.

[46] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "SEAL: attack mitigation for encrypted databases via adjustable leakage," in *USENIX Security 2020*.

[47] C. Zuo, S. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and L. Xu, "Forward and backward private DSSE for range queries," *IEEE Trans. Dependable Secur. Comput.*, 2022.

[48] B. Wang and X. Fan, "Search ranges efficiently and compatibly as keywords over encrypted data," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[49] F. Falzon, E. A. Markatou, Z. Espiritu, and R. Tamassia, "Range search over encrypted multi-attribute data," 2022.

[50] E. Shi, J. Bethencourt, T. H. Chan, D. X. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *2007 IEEE Symposium on Security and Privacy (S&P)*.

[51] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li, "Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index," ser. ASIA CCS 2014. Association for Computing Machinery.

[52] S. D. C. di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati, "Multi-dimensional indexes for point and range queries on outsourced encrypted data," in *IEEE Global Communications Conference, GLOBECOM*, 2021.

[53] J. Wang and S. S. M. Chow, "Forward and backward-secure range-searchable symmetric encryption," *Proceedings on Privacy Enhancing Technologies*, 2021.

[54] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS 2012*, 2012.

[55] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *ACM CCS 2015*, 2015, pp. 668–679.

[56] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *USENIX Security Symposium 2016*, 2016, pp. 707–720.

[57] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting leakage abuse attacks," in *NDSS 2020*.

[58] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *ACM CCS 2016*.

[59] S. Oya and F. Kerschbaum, "Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption," in *USENIX Security 2021*.

[60] F. Falzon, E. A. Markatou, Akshima, D. Cash, A. Rivkin, J. Stern, and R. Tamassia, "Full database reconstruction in two dimensions," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*.

[61] E. A. Markatou, F. Falzon, R. Tamassia, and W. Schor, "Reconstructing with less: Leakage abuse attacks in two dimensions," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*.

[62] P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson, "Pump up the volume: Practical database reconstruction from volume leakage on range queries," in *ACM CCS 2018*.

[63] P. Grubbs, M. S. Lacharité, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," in *IEEE SP 2019*.

[64] M. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *IEEE SP 2018*. IEEE Computer Society, 2018, pp. 297–314.

[65] Z. Gui, O. Johnson, and B. Warinschi, "Encrypted databases: New volume attacks against range queries," in *ACM CCS 2019*.

[66] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks," in *42nd IEEE Symposium on Security and Privacy, SP 2021*.

[67] E. A. Markatou and R. Tamassia, "Full database reconstruction with access and search pattern leakage," in *Information Security - 22nd International Conference*, 2019.

[68] K. Ren, Y. Guo, J. Li, X. Jia, C. Wang, Y. Zhou, S. Wang, N. Cao, and F. Li, "Hybridx: New hybrid index for volume-hiding range queries in data outsourcing services," *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*.

[69] Y. Chen, Q. Zheng, Z. Yan, and D. Liu, "Qshield: Protecting outsourced cloud data queries with multi-user access control based on sgx," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[70] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 2018)*.

[71] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[72] S. Kamara and T. Moataz, "Computationally volume-hiding structured encryption," in *EUROCRYPT 2019*.

[73] M. Ando and M. George, "On the cost of suppressing volume for encrypted multi-maps," *Proc. Priv. Enhancing Technol.*, 2022.

[74] S. Patel, G. Persiano, J. Y. Seo, and K. Yeo, "Efficient boolean search over encrypted data with reduced leakage," in *ASIACRYPT 2021*.

[75] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *ACM CCS 2018*, 2018, pp. 1038–1055.

[76] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *ACM CCS 2017*, 2017, pp. 1449–1463.

[77] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," *PoPETs*, vol. 2018, no. 1, pp. 5–20, 2018.

[78] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *ACM CCS 2017*, 2017, pp. 1465–1482.

Appendix

1. Dynamic SSE

A dynamic searchable symmetric encryption (SSE) scheme consists of a polynomial-time algorithm `Setup`

executed by the client, and protocols Search and Update executed jointly by the client and the server:

- $\texttt{Setup}(\lambda)$: A probabilistic algorithm that takes the security parameter λ. It outputs the tuple $(\mathsf{sk}, \mathsf{st}, \mathbf{EDB})$, where sk is the client's secret-key, st is the client's internal state, and **EDB** is an *empty* encrypted database.
- $\texttt{Update}(\mathsf{sk}, \mathsf{st}, \mathsf{op}, (\mathsf{id}, \mathsf{w}); \mathbf{EDB})$: A client-server protocol, where the client takes as input the secret-key sk, its state st, an operation $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$ and an identifier-keyword pair $(\mathsf{id}, \mathsf{w})$, while the server takes as input the encrypted database **EDB**. The protocol outputs a modified client state st' and a modified encrypted database $\mathbf{EDB}'$ so as to reflect the outcome of the addition/deletion operation.
- $\texttt{Search}(\mathsf{sk}, \mathsf{st}, q; \mathbf{EDB})$: A client-server protocol, where the client takes as input the secret-key sk, its state st and a query q, while the server takes as input the encrypted database **EDB**. At the end of the protocol, the client outputs $\mathbf{DB}(q)$.

In the above, we adopted the definition of dynamic SSE used by Chamani *et al.* [75]. There exist other definitions of dynamic SSE in the literature [76], [77] where the Update operation takes an entire file for addition/deletion, which is functionally equivalent to executing multiple addition/deletion operations on the relevant identifier/keyword pairs in our framework. Finally, we make the implicit assumption that upon obtaining the set of file identifiers corresponding to a query, the client performs an additional interaction with the server to actually retrieve the files with these identifiers.

Correctness. A dynamic SSE is said to be correct if for every database **DB** and for every query q, the Search protocol outputs $\mathbf{DB}(q)$ with all but negligible probability.

Security. The security of a dynamic SSE scheme is parameterized by a leakage function

$$\mathcal{L} = \left(\mathcal{L}^{\texttt{Setup}}, \mathcal{L}^{\texttt{Search}}, \mathcal{L}^{\texttt{Update}}\right),$$

where $\mathcal{L}^{\texttt{Setup}}$ encapsulates the leakage to an adversarial server during the setup phase, $\mathcal{L}^{\texttt{Search}}$ encapsulates the leakage to an adversarial server during each execution of the search protocol, and $\mathcal{L}^{\texttt{Update}}$ encapsulates the leakage to an adversarial server during each execution of the update protocol.

Informally, a dynamic SSE scheme is secure with respect to a leakage function $\mathcal{L}$ if the adversarial server provably learns no more information about **DB** other than that encapsulated by $\mathcal{L}$. Formally, a dynamic SSE scheme is said to be adaptively-secure with respect to a leakage function $\mathcal{L}$ if for any stateful PPT adversary $\mathcal{A}$ that issues a maximum of $Q = \mathsf{poly}(\lambda)$ queries, there exists a stateful probabilistic polynomial-time simulator $\textsc{Sim} = (\textsc{SimSetup}, \textsc{SimSearch}, \textsc{SimUpdate})$ such that the following holds:

$$\left|\Pr\left[\mathbf{Real}_{\mathcal{A}}^{\mathsf{Dy-SSE}}(\lambda, Q) = 1\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{A},\textsc{Sim}}^{\mathsf{Dy-SSE}}(\lambda, Q) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the "real" experiment $\mathbf{Real}^{\mathsf{Dy-SSE}}$ and the "ideal" experiment $\mathbf{Ideal}^{\mathsf{Dy-SSE}}$ are as described in [23].

Disjunctive Queries. We represent a disjunctive query over n distinct keywords $\mathsf{w}_1, \ldots, \mathsf{w}_n$ as $q = (\mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n)$ and define the set $\mathbf{DB}(q)$ as $\mathbf{DB}(q) = \cup_{i=1}^{n} \mathbf{DB}(\mathsf{w}_i)$.

Range Queries. We represent a d-dimensional range query $q = [a_1, b_1] \times \ldots \times [a_d, b_d]$ where $[a_i, b_i] \subseteq [1, |\mathcal{D}_i|]$ denotes the range in the i-th dimension.

2. TWINSSE: Overview

TWINSSE [20] is a fast and highly scalable SSE scheme with support for conjunctive, disjunctive and more general Boolean queries (in both CNF and DNF forms), that uses a conjunctive SSE scheme as a black box. The main contribution of the work is to support conjunctive, disjunctive and general Boolean queries with *linear* storage overhead that outperforms the existing SSE schemes [22] with similar query expressiveness. The core technical contribution of TWINSSE is generation of a "meta-database" that consists of specially designed "meta-keywords" (mkw), which is a disjunctive combination of specific keywords from the database (for more details on the construction of "meta-keywords" see [20]). The motivation for generating meta-keywords is to use a conjunctive SSE as a black box and support richer queries including conjunctions, disjunctions, and more general Boolean queries over encrypted data.

For example, a query of the form $q = \mathsf{w}_1 \vee \mathsf{w}_2 \vee \ldots \vee \mathsf{w}_\mathsf{n}$ is processed by TWINSSE as follows - The original disjunctive query q is transformed into a corresponding meta-query $q\mathsf{mkw} = \mathsf{mkw}_1 \wedge \mathsf{mkw}_2 \wedge \ldots \wedge \mathsf{mkw}_{\mathsf{n}'}$ (where $\mathsf{n}' \geq \mathsf{n}$) consisting of a conjunction of meta-keywords corresponding to the keywords present in q. The transformed meta-query is then given to the conjunctive SSE oracle. The result returned by the SSE oracle is a super set of the result corresponding to the original disjunctive query q. Thus, TWINSSE provides a consolidated framework by deploying a conjunctive SSE as a black box that provides support for general Boolean queries over encrypted database.

The meta-keywords scale linearly with the number of keywords in the original database. The storage overhead of TWINSSE hence scales linearly with the plaintext database size. This results in a significant improvement over the quadratic storage overheads incurred by state-of-the-art SSE [22] supporting disjunctive and Boolean queries. The search time complexity of TWINSSE depends upon the underlying conjunctive SSE. In [20] the authors show an instantiation of TWINSSE from OXT [27] ($\texttt{TWINSSE}_{\texttt{OXT}}$). The experimental results depict, $\texttt{TWINSSE}_{\texttt{OXT}}$ exhibits sub-linear search time complexity for both conjunctive and disjunctive queries, which makes it amenable to scale to large real-world databases.

3. GeneSSE: Proof of Correctness

The proof of correctness of GeneSSE follows from (1) correctness of Gen_mkw, (2) correctness of

GenMQuery [20] given that the frequency based bucketization has no effect on the correctness of TWINSSE, and (3) correctness of our modifications to ODXT [23]. We explain each in effect.

Correctness of `Gen_mkw`. Gen_mkw performs the following modifications to the original algorithm for generation of meta-database in [20]:

1. In [20], the set Δ is known in advance since [20] considers the static database model. In GeneSSE, Δ is constructed adaptively (line 3, Algorithm 2).
2. In [20], since the meta-database is generated one-time during SSE setup phase, meta-keywords for all buckets are generated at once. In GeneSSE, meta-keywords against the bucket of only the *updated* keyword (line 6, Algorithm 2) are generated/updated.

We note modifications (1) and (2) are correct w.r.t. correctness of meta-keyword generation. This follows from the arguments presented against Theorem 4.1 in [20]: given a partition on Δ as $\Delta_1 \cup \cdots \cup \Delta_{n_B}$ for n_B buckets, application of $\texttt{TWINSSE}_{\texttt{Basic}}$ on each of the buckets ensures correctness of meta-keyword generation. Note that each bucket is *independently* operated upon by $\texttt{TWINSSE}_{\texttt{Basic}}$ for meta-keyword generation.

Now in GeneSSE, Gen_mkw is invoked for each query, which contains an update operation for exactly *one* keyword w. Line 3 in Algorithm 2 serves to update the set Δ by adding the updated keyword w to Δ; this allows for precise calculation of the number of buckets n_B dynamically. In other words, modification (1) ensures correctness of n_B. Secondly, by design, the bucketization in GeneSSE ensures that w falls in exactly a *single* bucket in the meta-database. In other words, $n_B - 1$ buckets are untouched by the updates to w. Line 6 in Algorithm 2 serves to pick that *single* bucket in the meta-database and regenerate meta-keywords for it, while leaving other buckets untouched. Recall from Theorem 4.1 in [20] that meta-keyword generation is *independent* for each bucket. This directly implies the correctness of modification (2) in GeneSSE: updating bucket Δ_k (lines 7-11, Algorithm 2) has no impact on other $n_B - 1$ buckets present in the meta-database; there is hence no need for Gen_mkw to update these $n_B - 1$ buckets to ensure correctness.

The remaining argument of correctness thereafter follows directly from [20]; Appendix B: where a query of form $\mathbf{DB}(q_{\mathsf{mkw}})$ is correctly decomposed into sub-queries for each constituent bucket:

$$q_{\mathsf{mkw}} = \bigcup_{u \in [n_B]} \mathbf{DB}(q_{\mathsf{mkw},u})$$

Which, by structure of meta-keyword generation (which Gen_mkw does not change) finally leads to $\mathbf{DB}(q) \subseteq \mathbf{DB}(q_{mkw})$ with $\mid \mathbf{DB}(q) \mid = \eta \cdot \mid \mathbf{DB}(q_{mkw}) \mid$ for precision parameter η (see our experimental results in Section 4 for details on concrete instantiation of η).

Correctness of `GenMQuery`. The only change in GenMQuery when shifting from the static database model in [20] to the dynamic database model in GeneSSE is the strategy undertaken to sort the keywords in query q. In [20], this sorting is done based on keyword frequency; in GeneSSE, the ordering is based on keyword update frequency. We note from Theorem 4.1 in [20] (as well as our discussions in Section 2.1) that sorting/ordering of keywords is solely for concrete improvement in search efficiency, and does not play a role in ensuring correctness. Thus, GenMQuery is correct.

Correctness of GeneSSE.`Search` and GeneSSE.UPDATE. Algorithm 3 and Algorithm 5 denote procedures for updates and searches in GeneSSE. We now show that correctness of Algorithm 3 and Algorithm 5 follows from the structure of GeneSSE. Consider a disjunctive query of the form:

$$q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$$

By the correctness of Gen_mkw algorithm, it is converted into a corresponding meta-query of the form -

$$q_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$$

Once the meta-query is generated, the client identifies the meta-keyword with the least value of update count (UC). It generates the stokenList of TSet addresses using all updates of this meta-keyword. By the correctness of TSet routine [27], the correct set of $(\texttt{sval}, \alpha)$ is returned. For all such pairs in the set and all other meta-keywords q_{mkw}, the client generates a set of $\mathbf{xtoken}$ as -

$$\mathbf{xtoken}_{(i,p),l,j} = g^{F_p(K_X, \mathsf{mkw}_i^{k'} || \mathsf{w}_{k',p}) \cdot F_p(K_Z, \mathsf{mkw}_1^k || \mathsf{w}_{k,l} || j)}$$

where, $i = 2$ to $n; j = 1$ to $|\texttt{stokenList}|; (k, k') \in [n_{\mathsf{B}}]; p \leftarrow \#\mathsf{w} \in \mathsf{mkw}_i^{k'}; l \leftarrow \#\mathsf{w} \in \mathsf{mkw}_1^k$. For an $\alpha_j == F_p(K_Y, \mathsf{id} || \mathsf{op}) \cdot (F_p(K_Z, \mathsf{mkw} || \mathsf{w} || \texttt{UC}[\mathsf{mkw}]))^{-1}$ corresponding to mkw_1^k and a correct $\mathbf{xtoken}_{(i,p),l,j}$[14], $(\mathbf{xtoken}_{(i,p),l,j})^{\alpha_j}$ will compute a correct $\mathbf{xtag}$ that is present in the XSet. The correctness of the $\mathbf{xtag}$ computation is guaranteed by the structure of α and $\mathbf{xtoken}$. For all correctly matched $\mathbf{xtag}$, the $(\texttt{sval}, \alpha)$ values will be returned to the client and the corresponding ids will be retrieved by the client. The $\mathbf{DB}(q_{\mathsf{mkw}})$ thus obtained is a superset of the actual result of the disjunction, i.e. $\mathbf{DB}(q_{\mathsf{mkw}}) \subseteq \mathbf{DB}(q)$; this holds from the correctness of Gen_mkw, thus proving the correctness of GeneSSE.

4. GeneSSE: Security Analysis

We present an informal overview of the leakage profile of GeneSSE, followed by a detailed formal leakage profile analysis, transitively followed by a formal proof of security is present.

14. For a $\mathsf{w}_p^{k'} || \mathsf{mkw}_i^{k'}$ pair that matches $\mathsf{w}_l^k || \mathsf{mkw}_1^k$

4.1. Leakage profile of GeneSSE. Setup Leakage. Leakage from GeneSSE.Setup is similar to that of TWINSSE. The total size of the database (**DB**) is revealed, formally stated as -

$$|\mathbf{DB}| = \Sigma_{\mathsf{w}\in\Delta}|\mathbf{DB}(\mathsf{w})|$$

Update Leakage. Updates in GeneSSE.UPDATE are leakage free. This is a direct advantage of using ODXT: ODXT supports leakage-free updates. Only the TSet and XSet addresses are seen by the server that are randomly generated using a PRF.

Search Leakages. For a disjunctive query of the form $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$, GeneSSE.Search converts it into a corresponding conjunction of meta-keywords $q_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$, that is given to the server.

(i) *Result Pattern* (RP): Server learns the final result set, i.e. the document identifiers matching the conjunctive query of meta-kewords q_{mkw}.

$$\mathsf{RP}(q_{\mathsf{mkw}}) = \mathbf{DB}(q_{\mathsf{mkw}})$$

Note, that the result of the original disjunctive query q is a subset of q_{mkw} i.e., $\mathbf{DB}(q) \subseteq \mathbf{DB}(q_{\mathsf{mkw}})$.

(ii) *Size Pattern* (SP): Since the updates occur w.r.t to the meta-keywords, the server learns the number of updates pertaining to the least updated meta-keyword (mkw_1 w.l.o.g), termed as the "sterm" henceforth [15], as well as the time stamp for each update.

(iii) *Equality Pattern* (EP): The server learns the common stermfor two or more conjunctive queries. This is because for the same sterm in two queries, the client generates the same stokens (see Algorithm 5) corresponding to mkw_1.

(iv) *x-term Leakage*: For each update operation $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_1))$ involving the sterm mkw_1, the server learns the total number of update operations of the form $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_i))$ for each xterm $\mathsf{mkw}_i \in \{\mathsf{mkw}_2, \ldots, \mathsf{mkw}_n\}$ for a conjunctive query $q = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$, along with the corresponding time stamp for each update.

(v) *Common x-term Leakage*: The server learns if two queries with distinct sterms mkw_1 and mkw'_1 share a common xterm (mkw_i), provided that the update corresponding to mkw_1 and mkw'_1 involves atleast one common $(\mathsf{id}_j, \mathsf{w}_l)$ pair (where $\mathsf{w}_l \in (\mathsf{mkw}_1 \cap \mathsf{mkw}'_1)$). This leakage occurs because the $\mathbf{xtag}$ corresponding to the same $(\mathsf{id}_j, \mathsf{w}_l)$ pair.

Improvements over ODXT. Since GeneSSE is proposed as a dynamic disjunctive SSE where we specifically leverage the disjunctive operation of the unerlying TWINSSE framework, **some of the leakages specific to the conjunctive queries in ODXT are rendered inconsequential w.r.t. the proposed disjunctive framework**. We explain this now.

15. This is by the design choice of OXT [27],ODXT [23], the the TSet addresses are generated corresponding to the least frequent term/ the term with least update count in the query. We note that selection of "sterm" is to achieve sub-linear search complexity

Size Pattern: Since the updates occur w.r.t to the meta-keywords in GeneSSE, the **server does not essentially learn any information regarding the underlying keyword in the original disjunctive query**. Therefore, the update corresponding to an sterm does not imply the updates corresponding to any specific keyword in the database.

Equality Pattern Leakage: As meta-keywords are a disjunctive composition of multiple keywords, the **server cannot infer any correlation between the actual keywords in the query and the** sterm **it observes over multiple iterations**. That is, the same sterm for two queries in in this case does not imply the same keyword is being queried in the original disjunction.

x-term Leakage: For every update of the form $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_1))$ involving the sterm mkw_1, the server learns the total number of update operations of the form $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_i))$ for each xterm $\mathsf{mkw}_i \in \{\mathsf{mkw}_2, \ldots, \mathsf{mkw}_n\}$, along with the corresponding time stamp for each update. **The server, however, does not learn the number of updates corresponding to actual keywords in the disjunctive query**. Also, updates corresponding to a meta-keyword do not reveal update history of any specific keyword. Each meta-keyword mkw is a disjunctive composition of specific keywords; therefore, an update pertaining to any $\mathsf{w} \in \mathsf{mkw}$ will lead to an update of mkw. Since the specific keywords constituent in a mkw is not known to the server, it is not possible for it to track the update history of any specific keyword from the update history of mkw. For an update of the form $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{w}_1))$, it is converted to the corresponding meta-keyword update that w corresponds to.

Common x-term Leakage: The server learns if two queries with distinct sterms mkw_1 and mkw'_1 share a common xterm (mkw_i), provided that the update corresponding to mkw_1 and mkw'_1 involves atleast one common $(\mathsf{id}_j, \mathsf{w}_l)$ pair (where $\mathsf{w}_l \in (\mathsf{mkw}_1 \cap \mathsf{mkw}'_1)$). This leakage occurs because the $\mathbf{xtag}$ corresponding to the same $(\mathsf{id}_j, \mathsf{w}_l)$ pair. **This will also not leak anything about the keywords in the query, as the** sterm **or** xterm **all correspond to meta-keywords**.

4.2. Formal Leakage profile of GeneSSE. In this section, we give a formal analysis of the leakage profile of GeneSSE and prove its forward and backward privacy. A dynamic disjunctive SSE scheme is forward and backward private if: (i) an update operation reveals no additional information about a disjunctive search operation that took place at an earlier time, and (ii) a search operation on a disjunctive query $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$ reveals no information about certain deletion operations on w_n that took place at an earlier time. We formally establish below that GeneSSE achieves this notion of forward and backward privacy.

Let Q be a list with the following types of entries:

(i) (t, w): Search query on keyword w at timestamp t.

(ii) Update query $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$ on the identifier-keyword pair $(\mathsf{id}, \mathsf{w})$ at time stamp t.

Output Leakages. For any keyword w, we define $\mathtt{TimeDB}(\mathsf{w})$ to be the function that returns the list of all file

identifiers containing w that have not been deleted yet, along with their respective time stamps of insertion. Formally -

$$\begin{aligned}\texttt{TimeDB}(\mathsf{w}) \quad = \quad & \{(t,\mathsf{id})|(t,\mathsf{add},(\mathsf{id},\mathsf{w}))\in Q \\ & \textit{and} \quad \forall t' : (t',\mathsf{del},(\mathsf{id},\mathsf{w}))\notin Q\}\end{aligned}$$

For a conjunction of meta-keywords $q_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$, $\texttt{TimeDB}(\mathsf{q}_{\mathsf{mkw}})$ is defined as -

$$\begin{aligned}\texttt{TimeDB}(\mathsf{q}_{\mathsf{mkw}}) = \{(\{t_i\}_{i\in[n]},\mathsf{id})|(t_i,\mathsf{add},(\mathsf{id},\mathsf{mkw}_i))\in Q \\ \textit{and} \quad \forall t' : (t',\mathsf{del},(\mathsf{id},\mathsf{mkw}_i))\notin Q\}\end{aligned}$$

In other words, $\texttt{TimeDB}(\mathsf{q}_{\mathsf{mkw}})$ returns the list of identifiers corresponding to documents containing $q_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$ that have not yet been deleted, along with their respective time stamps of insertion. Thus, $\texttt{TimeDB}(\mathsf{q}_{\mathsf{mkw}})$ captures the output leakage for q_{mkw}.

Note. In case of GeneSSE for a disjunctive query $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$, $\mathbf{DB}(q)$ is not actually computed by the server, instead it work on meta-keyowrds a and returns $\mathbf{DB}(q_{\mathsf{mkw}})$. Thus $\texttt{TimeDB}(\mathsf{q})$ does not capture the exact list of identifiers on $\mathbf{DB}(q)$. We can thus say, that the exact time stamps corresponding to the insertion of a particular keyword is obfuscated in GeneSSE. Thus, $\texttt{TimeDB}(\mathsf{q})$ for any disjunctive query $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$ is essentially $\perp$ -

$$\texttt{TimeDB}(\mathsf{q}) = \perp$$

x-term Leakages. We define $\mathsf{Upd}(\mathsf{w}_1,\mathsf{w}_2)$ for any pair of keywords $(\mathsf{w}_1,\mathsf{w}_2)$ as follows (similar as ODXT):

$$\begin{aligned}\mathsf{Upd}(\mathsf{w}_1,\mathsf{w}_2) = \{(t_1,t_2)|\exists(\mathsf{op},\mathsf{id}) : (t_1,\mathsf{op},(\mathsf{id},\mathsf{w}_1))\in Q \\ \textit{and} \quad (t_2,\mathsf{op},(\mathsf{id},\mathsf{w}_2))\in Q\}\end{aligned}$$

Thus, $\mathsf{Upd}(\mathsf{w}_1,\mathsf{w}_2)$ returns the time stamps of all update operations on w_1 and w_2 that involve the same document identifier. This however is not applicable for GeneSSE because for every update of a keyword w the corresponding meta-keywords in which w is present are updated. The set of meta-keywords for both w_1 and w_2 is might intersect but will never be identical by its structure. Therefore even if the id corresponding to the update of w_1 and w_2 is same, the timestamp corresponding to the updates cannot be exactly inferred because of the meta-keyword updates that actually happen at the server. Thus $\mathsf{Upd}(q)$ for a disjunctive query $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$, which is transformed into $q_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$ has the following $\mathsf{Upd}(\mathsf{mkw})$ (we overload notation for Upd and use its definition interchangeable for keywords and meta-keywords):

$$\mathsf{Upd}(q_{mkw}) = \mathsf{Upd}(\mathsf{mkw}_1) \cup \left(\bigcup_{i=2}^{n} \mathsf{Upd}(\mathsf{mkw}_1,\mathsf{mkw}_i)\right)$$

Formally the leakage profile of GeneSSE is defined as follows -

$$\mathcal{L}_{\texttt{GeneSSE}} = (\mathcal{L}^{\texttt{Setup}}_{\texttt{GeneSSE}}, \mathcal{L}^{\mathsf{Upd}}_{\texttt{GeneSSE}}, \mathcal{L}^{\texttt{Search}}_{\texttt{GeneSSE}})$$

such that -

(i) $\mathcal{L}^{\texttt{Setup}}_{\texttt{GeneSSE}} = \perp$

(ii) $\mathcal{L}^{\mathsf{Upd}}_{\texttt{GeneSSE}}(\mathsf{op},(\mathsf{id},\mathsf{w})) = \perp$

(iii) $\mathcal{L}^{\texttt{Search}}_{\texttt{GeneSSE}}(q) = (\texttt{TimeDB}(q_{mkw}), \mathsf{Upd}(q_{mkw}))$

Finally we prove the main theorem for GeneSSE.

4.3. Proof of Theorem 2. We prove Theorem 2 via a sequence of experiments between a challenger and an adversary, where the first experiment is identical to the real experiment $\mathbf{Real}^{\mathsf{Dy-SSE}}(\lambda,Q)$, while the final experiment is identical to the simulation experiment $\textsc{Sim}^{\mathsf{Dy-SSE}}(\lambda,Q)$. We establish formally that the view of the adversary $\mathcal{A}$ in each pair of consecutive experiments is computationally indistinguishable.

Experiment-0. This experiment is identical to the real experiment $\mathbf{Real}^{\mathsf{Dy-SSE}}(\lambda,Q)$. The challenger generates the transcript corresponding to each update operation using the GeneSSE.Update algorithm and the transcript corresponding to each disjunctive search using the GeneSSE.Search algorithm.

Experiment-1. This experiment is identical to Experiment-0, except that when generating the transcripts for each update and disjunctive search (conjunctive search on mkw), the challenger replaces each PRF evaluation of the form $F(K_T,\cdot)$ with a function evaluation of the form $G_T(\cdot)$, where the function G_T is uniformly sampled from the set of all functions that map λ-bit strings onto λ-bit strings.

Lemma A.1. *Assuming that F is a secure PRF, the view of the adversary $\mathcal{A}$ in Experiment-1 is computationally indistinguishable from the view of the adversary $\mathcal{A}$ in Experiment-0.*

Proof. Let $\mathcal{B}_1$ be a probabilistic polynomial-time algorithm that can distinguish between the views of the adversary $\mathcal{A}$ in Experiment-0 and Experiment-1. Then $\mathcal{B}_1$ can be used to design a probabilistic polynomial-time adversary $\mathcal{B}'_1$ that can distinguish a set of PRF evaluations of the form $F(K_T,\cdot)$ from a set of function evaluations of the form $G_T(\cdot)$, where the function G_T is uniformly sampled from the set of all functions that map λ-bit strings onto λ-bit strings. By definition, $\mathcal{B}'_1$ breaks the pseudorandomness of PRF F, which is a contradiction. □

Experiment-2a. This experiment is identical to Experiment-1, except that when generating the transcripts for each update and conjunctive search operation, the challenger replaces each PRF evaluation of the form $F_p(K_X,\cdot)$ with a function evaluation of the form $G_X(\cdot)$, where the function G_X is uniformly sampled from the set of all functions that map λ-bit strings onto elements in $\mathbb{Z}^*_p$.

Lemma A.2. *Assuming that F_p is a secure PRF, the view of the adversary $\mathcal{A}$ in Experiment-2a is computationally indistinguishable from the view of the adversary $\mathcal{A}$ in Experiment-1.*

Proof. Let $\mathcal{B}_2$ be a probabilistic polynomial-time algorithm that can distinguish between the views of the adversary $\mathcal{A}$ in

Experiment-1 and Experiment-2a. Then $\mathcal{B}_2$ can be used to design a probabilistic polynomial-time adversary $\mathcal{B}_2'$ that can distinguish a set of PRF evaluations of the form $F_p(K_X, \cdot)$ from a set of function evaluations of the form $G_X(\cdot)$, where the function G_X is uniformly sampled from the set of all functions that map λ-bit strings onto elements in $\mathbb{Z}_p^*$. By definition, $\mathcal{B}_2'$ breaks the pseudorandomness of F_p, which is a contradiction. □

Experiment-2b. This experiment is identical to Experiment-2a, except that when generating the transcripts for each update and disjunctive search (conjunctive search of mkw) operation, the challenger replaces each PRF evaluation of the form $F_p(K_Y, \cdot)$ with a function evaluation of the form $G_Y(\cdot)$, where the function G_Y is uniformly sampled from the set of all functions that map λ-bit strings onto elements in $\mathbb{Z}_p^*$.

Lemma A.3. *Assuming that F_p is a secure PRF, the view of the adversary $\mathcal{A}$ in Experiment-2b is computationally indistinguishable from the view of the adversary $\mathcal{A}$ in Experiment-2a.*

Proof. The proof of this lemma is identical to the proof of Lemma A.2. □

Experiment-2c. This experiment is identical to Experiment-2b, except that when generating the transcripts for each update and disjunctive search (conjunctive search of mkw) operation, the challenger replaces each PRF evaluation of the form $Fp(K_Z, \cdot)$ with a function evaluation of the form $G_Z(\cdot)$, where the function G_Z is uniformly sampled from the set of all functions that map λ-bit strings onto elements in $\mathbb{Z}_p^*$.

Lemma A.4. *Assuming that F_p is a secure PRF, the view of the adversary $\mathcal{A}$ in Experiment-2c is computationally indistinguishable from the view of the adversary $\mathcal{A}$ in Experiment-2b.*

Proof. The proof of this lemma is identical to the proof of Lemma A.2. □

Experiment-3. This experiment is identical to Experiment-2c, except that when generating the transcripts for each disjunctive search operation, the challenger changes the manner in which each $\mathbf{xtoken}$ value is generated. More specifically, for a disjunctive query $q = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$ which is converted into a corresponding meta-query $q_{\mathsf{mkw}} = \mathsf{mkw}_1 \wedge \ldots \wedge \mathsf{mkw}_n$, the challenger first looks up the history of update queries made by the adversary $\mathcal{A}$ to retrieve the set of update operations $\{(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_1))\}$ involving the sterm mkw_1. Next, for each keyword mkw_i in the conjunction and each update operation $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_1))$, it locally computes the corresponding TSet blinding factor $\alpha_{i,j}$ and the corresponding cross-tag $\mathbf{xtag}_{i,j}$ exactly as in the real GeneSSE scheme, and generates the corresponding $\mathbf{xtoken}_{i,j}$ as: $\mathbf{xtoken}_{i,j} = \mathbf{xtag}_{i,j}^{1/\alpha_{i,j}}$. For the *padded* $\mathbf{xtoken}$s (Section 2.2), this experiment randomly samples $\mathbf{xtag}$ from its appropriate domain, $\beta \leftarrow Z_p^*$, and sets $\mathbf{xtoken} = \mathbf{xtag}_{i,j}^{1/\alpha_{i,j}} = (g^\beta)^{1/\alpha_{i,j}}$, where g is the generator for the group $\mathcal{G}$.

Lemma A.5. *The view of the adversary $\mathcal{A}$ in Experiment-3 is identical to the view of the adversary $\mathcal{A}$ in Experiment-2c.*

Proof. The proof of this lemma follows immediately because the distribution of each $\mathbf{xtoken}$ value in Experiment-2c is identical to the distribution of each $\mathbf{xtoken}$ value in Experiment-3. In Experiment-2c for any triplet $(\alpha_{i,j}, \mathbf{xtag}_{i,j}, \mathbf{xtoken}_{i,j})$, we have $\mathbf{xtag}_{i,j} = \mathbf{xtoken}_{i,j}^{\alpha_{i,j}}$, i.e., $\mathbf{xtoken}_{i,j} = \mathbf{xtag}_{i,j}^{1/\alpha_{i,j}}$, which is identical to Experiment-3. The joint distribution of actual $\mathbf{xtoken}$s and padded $\mathbf{xtoken}$s is independent and random in $\mathbb{Z}_p^*$, considering DDH assumption holds in $\mathcal{G}$ Thus the view of adversary $\mathcal{A}$ in Experiment-3 is identical to the view of the adversary $\mathcal{A}$ in Experiment-2c. □

Experiment-4. This experiment is identical to Experiment-3, except that when generating the transcripts for each update operation, the challenger changes the manner in which each pre-computed blinding factor value is generated i.e., instead of computing α as in Experiment-3, it simply samples $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$. For the *padded* $\mathbf{xtoken}$s (Section 2.2), this experiment randomly samples $\mathbf{xtag}$ from its appropriate domain, $\beta \leftarrow Z_p^*$, and sets $\mathbf{xtoken} = \mathbf{xtag}_{i,j}^{1/\alpha_{i,j}} = (g^\beta)^{1/\alpha_{i,j}}$, where g is the generator for the group $\mathcal{G}$.

Lemma A.6. *The view of adversary $\mathcal{A}$ in Experiment-4 is statistically indistinguishable from the view of adversary $\mathcal{A}$ in Experiment-3.*

Proof. Function G_Z is uniformly randomly sampled from the set of all functions that map λ-bit strings onto elements in $\mathbb{Z}_p^*$. Also due to the changes made in Experiment-3 to the generation of $\mathbf{xtoken}$ values, the function G_Z is only evaluated once during each update operation (to generate the blinding factor to be stored in the TSet), and is never re-evaluated during any of the conjunctive search (of mkw). Finally, the function G_Z is never evaluated on the same input in two different update operations, and each evaluation of G_Z is independent of the random coins used by the challenger in the rest of the experiment.

In Experiment-3, each pre-computed blinding factor α value in $\mathbb{Z}_p^*$ is generated as the product of an evaluation of G_Y in $\mathbb{Z}_p^*$ with the inverse of an evaluation of G_Z in $\mathbb{Z}_p^*$. Finally, The joint distribution of actual $\mathbf{xtoken}$s and padded $\mathbf{xtoken}$s is independent and random in $\mathbb{Z}_p^*$, considering DDH assumption holds in $\mathcal{G}$. These facts, combined with the aforementioned observations, immediately imply that distribution of each value of α in Experiment-4 is statistically indistinguishable from that in Experiment-3. □

Experiment-5. This experiment is identical to Experiment-4, except that when generating the transcripts for each update operation, the challenger changes the manner in which each $\mathbf{xtag}$ value is generated. Instead of computing each $\mathbf{xtag}$ value as in Experiment-4, it simply samples $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathbf{xtag} = g^\gamma$, where g is the generator for the group $\mathcal{G}$.

Lemma A.7. *Assuming that the DDH assumption holds in the group $\mathcal{G}$, the view of the adversary $\mathcal{A}$ in Experiment-5 is computationally indistinguishable from the view of the adversary $\mathcal{A}$ in Experiment-4.*

The proof of this lemma follows from the fact that extended DDH assumption holds in the group $\mathcal{G}$ in a similar manner as done in ODXT [23].

Experiment-6. This experiment is identical to Experiment-5 except that when generating the transcript for each update operation, the challenger replaces each function evaluation of the form $G_T(\mathsf{mkw}||cnt||b)$ for $b \in \{0,1\}$ with a function evaluation of the form $G_T(t)$, where t is the time stamp at which the update operation is executed. Similarly, when generating the transcript for each conjunctive search operation (of mkw), the challenger replaces each function evaluation of the form $GT(\mathsf{mkw}||cnt||0)$ with a function evaluation of the form $G_T(t)$, where t is the time stamp corresponding to the operation that this evaluation is meant to address in the TSet.

Lemma A.8. *The view of the adversary $\mathcal{A}$ in Experiment-6 is identical to the view of the adversary $\mathcal{A}$ in Experiment-5.*

Proof. In Experiment-5, the function G_T is never evaluated on the same input at two different time stamps due to the presence of the monotonically increasing counter variable. In addition, the function G_T is uniformly randomly sampled from the set of all functions that map λ-bit strings onto λ-bit strings. This immediately implies that the distribution of G_T evaluations in Experiments 5 and 6 are identical from the point of view of the adversary $\mathcal{A}$. □

Experiment-7. This experiment is identical to Experiment-6 except that we now replace the challenger with a simulator SIM that does not have access to the actual queries made by the adversary $\mathcal{A}$. Instead, SIM only has access to the following leakage profile for each update/conjunctive query issued by the adversary $\mathcal{A}$.

Update Leakages: SIM has access to an empty update leakage function, i.e., it gains no information about any of the update queries issued by $\mathcal{A}$. It uses the time stamp of the update query to generate the TSet (address, value) pair, as done by the challenger in Experiment-6. It generates a uniformly random blinding factor α, as done by the challenger in Experiment-6. Finally, it samples $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathbf{xtag} = g^{\gamma}$, where g is the generator for the group $\mathcal{G}$, as done by the challenger in Experiment-6.

Search Leakages. SIM has access to the following search leakages.

Size Pattern: SIM learns the number of update operations involving the sterm mkw_1, as well as the time stamp for each such operation, which it uses to simulate the set of stoken values using the function G_X as done by the challenger in Experiment-6.

x-term Leakage: For each update operation $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_1))$ involving the sterm mkw_1, SIM learns the total number of updates of the form $(\mathsf{op}_j, (\mathsf{id}_j, \mathsf{mkw}_i))$ for each xterm $\mathsf{mkw}_i \in \{\mathsf{mkw}_2, \ldots, \mathsf{mkw}_n\}$, as well the corresponding time stamp for each such operation. It uses this information to determine the corresponding $\mathbf{xtag}_{i,j}$ value and the corresponding blinding factor $\alpha_{i,j}$ and then generate $\mathbf{xtoken}_{i,j}$ as: $\mathbf{xtoken}_{i,j} = \mathbf{xtag}_{i,j}^{1/\alpha_{i,j}}$ as done by challenger in Experiment-6.

Result Pattern: SIM learns the final set of document identifiers in the conjunction of meta-keywords, and uses these to retrieve the corresponding documents.

Equality Pattern: SIM learns if two (or more) conjunctive queries have the same sterm mkw_1, and generates the stoken values to be consistent across these queries by evaluating the function G_X on the same set of time stamps.

Common x-term Leakage: SIM learns if two queries with (possibly distinct) sterms mkw_1 and mkw_1' share a common xterm mkw_i, provided that the update histories for mkw_1 and mkw_1' involve at least one common document identifier id_j. When processing these queries, SIM makes sure to generate the same cross-tag $\mathbf{xtag}_{i,j}$.

Lemma A.9. *The view of the adversary $\mathcal{A}$ in Experiment-7 is identical to the view of the adversary $\mathcal{A}$ in Experiment-6.*

Proof. The proof of this lemma follows immediately from the fact that the transcripts generated by SIM corresponding to each update and conjunctive meta-keyword query issued by the adversary $\mathcal{A}$ are identical to the corresponding transcripts generated by the challenger in Experiment-6. □

Experiment-8. This experiment is identical to Experiment-7 except that we now replace the simulator SIM with a simulator SIM$'$ that has access to a leakage function $\mathcal{L} = (\mathcal{L}^{\mathtt{Setup}}, \mathcal{L}^{\mathsf{Upd}}, \mathcal{L}^{\mathtt{Search}})$, such that -

$$\mathcal{L}^{\mathsf{Upd}}(\mathsf{op}, (\mathsf{id}, \mathsf{mkw})) = \perp$$

$$\mathcal{L}^{\mathtt{Search}}(q_{\mathsf{mkw}}) = (\mathtt{TimeDB}(q_{\mathsf{mkw}}), \mathsf{Upd}(q_{\mathsf{mkw}}))$$

The simulator SIM$'$ runs simulator SIM in Experiment-7 as a sub-routine when interacting with the adversary $\mathcal{A}$.

Lemma A.10. *The view of the adversary $\mathcal{A}$ in Experiment-8 is identical to the view of the adversary $\mathcal{A}$ in Experiment-7.*

The proof of this lemma follows in the similar manner as in ODXT. We mark here that the entire leakage from which SIM generates the transcripts is related to the meta-keywords. No leakage w.r.t to the keywords is captured by the leakage profile of GeneSSE.
This completes the proof of Theorem 2.

5. Forward Privacy of GeneSSE

We formally describe the forward privacy guarantees of GeneSSE in this section. According to the formal definition introduced by [78], a dynamic disjunctive SSE scheme is adaptively secure with respect to a leakage profile

$$\mathcal{L} = (\mathcal{L}^{\mathtt{Setup}}, \mathcal{L}^{\mathsf{Upd}}, \mathcal{L}^{\mathtt{Search}})$$

is said to be adaptively *forward private* if there exists a stateless function $\mathcal{L}'$ such that for any arbitrary triplet $(\mathsf{op}, \mathsf{id}, \mathsf{w})$, we have

$$\mathcal{L}^{\mathsf{Upd}}(\mathsf{op}, (\mathsf{id}, \mathsf{w})) = \mathcal{L}'(\mathsf{op}, \mathsf{id})$$

This implies that an update operation computationally hides the underlying keyword w, therefore, it cannot be correlated with any previous search query involving w by a computationally bounded adversary. Since $\mathcal{L}^{\text{UPDATE}}_{\texttt{GeneSSE}} = \perp$, an update operation in GeneSSE hides not only the underlying keyword w, but also the identifier id and the operation op. In other words, the following is a natural corollary of Theorem-2:

Corollary 1 (Forward Privacy of GeneSSE)**.** *Assuming that F and F_p are secure PRFs and the decisional Diffie-Hellman assumption holds over the group $\mathcal{G}$,* GeneSSE *is adaptively forward private.*

6. Backward Privacy of GeneSSE

We formally describe the backward privacy of GeneSSE. According to the formal definition introduced by Bost et al. [78], a dynamic SSE scheme that supports *single* keyword searches and is adaptively secure with respect to some leakage function $\mathcal{L} = (\mathcal{L}^{\texttt{Setup}}, \mathcal{L}^{\mathsf{Upd}}, \mathcal{L}^{\texttt{Search}})$ is adaptively *Type-I backward private* if there exist stateless functions $\mathcal{L}'$ and $\mathcal{L}''$ such that for any $(\mathsf{op}, \mathsf{idw})$, we have

$$\mathcal{L}^{\mathsf{Upd}}(\mathsf{op}, (\mathsf{id}, \mathsf{w})) = \mathcal{L}''(\mathsf{op}, \mathsf{id}) \quad \text{and}$$

$$\mathcal{L}^{\texttt{Search}}(\mathsf{w}) = \mathcal{L}'''(\mathbf{DB}(\mathsf{w}), \texttt{UC}(\mathsf{w}))$$

We have -

$$\mathcal{L}^{\mathsf{Upd}}_{\texttt{GeneSSE}}(\mathsf{op}, (\mathsf{id}, \mathsf{mkw})) = \perp,$$

$$\mathcal{L}^{\texttt{Search}}_{\texttt{GeneSSE}}(q_{mkw}) = \{\texttt{TimeDB}(q_{mkw}), \mathsf{Upd}(q_{mkw})\}$$

for any disjunctive query q. This is a natural generalization of the aforementioned leakage profile of Type-I backward privacy from the setting of a single keyword searches to our setting of disjunctive searches[16]. Note that the structure of meta-keywords ensures no correlations about $\texttt{UC}(\mathsf{w})$ can be drawn from $\mathsf{Upd}(q_{mkw})$. Hence, a natural corollary of Theorem-2 states -

Corollary 2 (Backward Privacy of GeneSSE)**.** *Assuming that F and F_p are secure PRFs and the decisional Diffie-Hellman assumption holds over the group $\mathcal{G}$,* GeneSSE *is adaptively Type-I backward private.*

16. Similar generalization have also been used to prove backward privacy guarantees of dynamic SSE scheme previously [23]

7. RGeneSSE Security Analysis

In this section we provide a simulation-based proof approach for RGeneSSE. We assume that GeneSSE is an adaptively secure dynamic disjunctive SSE with respect to the following leakage profile -

$$\mathcal{L}_{\texttt{GeneSSE}} = (\mathcal{L}^{\texttt{Setup}}_{\texttt{GeneSSE}}, \mathcal{L}^{\mathsf{Upd}}_{\texttt{GeneSSE}}, \mathcal{L}^{\texttt{Search}}_{\texttt{GeneSSE}})$$

We define the leakage profile of RGeneSSEas follows -

$$\mathcal{L}_{\texttt{RGeneSSE}} = (\mathcal{L}^{\texttt{Setup}}_{\texttt{RGeneSSE}}, \mathcal{L}^{\mathsf{Upd}}_{\texttt{RGeneSSE}}, \mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}})$$

where,

$$\mathcal{L}^{\texttt{Setup}}_{\texttt{RGeneSSE}}(\mathbf{DB}) = \mathcal{L}^{\texttt{Setup}}_{\texttt{GeneSSE}}(\mathbf{DB})$$

$$\mathcal{L}^{\mathsf{Upd}}_{\texttt{RGeneSSE}}(\mathsf{op}, (\mathsf{id}, \mathsf{w})) = \mathcal{L}^{\mathsf{Upd}}_{\texttt{GeneSSE}}(\mathsf{op}, (\mathsf{id}, \mathsf{w}))$$

and,

$$\mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\texttt{range}}) = \mathcal{L}^{\texttt{Search}}_{\texttt{GeneSSE}}(q_{\texttt{dis}})$$

where $q_{\texttt{range}}$ is a range query of the form $q_{\texttt{range}} = [a_i, b_i]$ where $[a_i, b_i] \subseteq [1, |\mathcal{D}_i|]$ denotes the range in the i-th dimension and $q_{\texttt{dis}}$ is a disjunctive query of the form $q_{\texttt{dis}} = \mathsf{w}_1 \vee \ldots \vee \mathsf{w}_n$.

We show that RGeneSSEis secure against an adaptive semi-honest adversary $\mathcal{A}$, which has access to leakages from $\mathcal{L}_{\texttt{RGeneSSE}}$. We build a simulator SIM that generates **EDB** by $\mathcal{L}_{\texttt{RGeneSSE.Update}}$, and generates transcripts τ_i for query $q_{\texttt{range}}$ over **EDB** from the search leakages, $\mathcal{L}_{\texttt{RGeneSSE.search}}$. SIM has inputs from $\mathcal{L}_{\texttt{RGeneSSE}}$ only. To prove the security of RGeneSSE, we extend the simulator of the underlying forward and backward private GeneSSE, and explain how it handles simulation for range queries.

Simulating RGeneSSE**.Setup:** The Setup leakage of RGeneSSEleaks no information (similar to the Setup leakage of GeneSSE):

$$\mathcal{L}^{\texttt{Setup}}_{\texttt{RGeneSSE}} = \mathcal{L}^{\texttt{Setup}}_{\texttt{GeneSSE}} = \perp$$

Simulating RGeneSSE**.Update:** The UPDATE leakage of RGeneSSEis similar to the UPDATE leakage of GeneSSE. The simulator outputs its version of **EDB** according to the simulation process of GeneSSE (we proved that GeneSSE is provably simulation secure in Appendix D.3).

$$\begin{aligned}\mathbf{EDB}_{\text{SIM}} &= \text{SIM}^{\mathsf{Upd}}_{\texttt{RGeneSSE}}(\mathbf{DB}) \\ &= \text{SIM}^{\mathsf{Upd}}_{\texttt{GeneSSE}}(\mathbf{DB}) \\ &= \text{SIM}^{\mathsf{Upd}}_{\texttt{GeneSSE}}(\mathbf{DB}, n')\end{aligned}$$

where n' (bucket size) is a public parameter. Since, GeneSSE is proven simulation secure, it follows from the simulation security guarantee of GeneSSE that $\mathbf{EDB}_{\text{SIM}}$ is

indistinguishable from the one generated in the real experiment.

Simulating RGeneSSE**.Search:** A range query $q_{\texttt{range}}$ is transformed into a disjunctive query $q_{\texttt{dis}}$ from the output of the $\texttt{BRC}_{RT}$ algorithm (as shown in Algorithm 10). The covering set of nodes returned by $\texttt{BRC}_{RT}$ is used to formulate the disjunctive query. We argue that the adversary $\mathcal{A}$ does not gain any information about the original keywords in a disjunctive query with this simulation experiment. The distribution of **DB** for original keywords (hence, also for **EDB**) is abstracted by the meta-keywords. With $\mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}$ that entails the leakages from GeneSSE.Search or, $\mathcal{L}^{\texttt{Search}}_{\texttt{GeneSSE}}$, SIM invokes the simulator of GeneSSE for simulating the search on the disjunction of the nodes returned by $\texttt{BRC}_{RT}$. Since, GeneSSE.Search invoked in RGeneSSE.Search (Algorithm 10) executes over meta-keyword only, this leakage is expressed in the context of meta-keywords as below.

$$\mathcal{L}'_{\texttt{GeneSSE}} = \mathcal{L}_{\texttt{GeneSSE}}(\{\mathsf{mkw}_i\})$$

With this leakage information of GeneSSE, the search leakage of RGeneSSE can be expressed as below -

$$\begin{aligned}\mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\texttt{range}}) &= \mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\texttt{dis}}) \\ &= \mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\mathsf{mkw},k})_{k\in[n_{\mathsf{B}}]} \\ &= \{\mathcal{L}'_{\texttt{GeneSSE}}, n'\}\end{aligned}$$

The parameters n_{B} and n' are derived from N (number of keywords), which is available during setup. Therefore, the search leakage of RGeneSSE is same as the underlying GeneSSE, which can be summarized as below.

$$\begin{aligned}\mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\texttt{range}}) &= \mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\texttt{dis}}) \\ &= \mathcal{L}^{\texttt{Search}}_{\texttt{RGeneSSE}}(q_{\mathsf{mkw},k})_{k\in[n_{\mathsf{B}}]} \\ &= \mathcal{L}'_{\texttt{GeneSSE}}\end{aligned}$$

This same leakage profile for search in RGeneSSE and GeneSSE in the context of meta-keywords ensures that no additional information is leaked beyond GeneSSE leakage.

8. Forward and Backward Privacy of RGeneSSE

Since GeneSSE is proven to be adaptively forward private and Type-II backward private in Appendix E and F, the forward and backward privacy guarantees of RGeneSSEfollows directly from the following corollary.

Corollary 3 (Forward Privacy of RGeneSSE)**.** *Assuming* GeneSSE *is adaptively forward private dynamic disjunctive SSE,* RGeneSSE*is an adaptively forward private dynamic range SSE scheme.*

Corollary 4 (Backward Privacy of RGeneSSE)**.** *Assuming* GeneSSE *is Type-I backward private dynamic disjunctive SSE,* RGeneSSE*is a Type-I backward private dynamic range SSE scheme.*