Efficient Boolean Search over Encrypted Data with Reduced Leakage

Sarvar Patel[1], Giuseppe Persiano[1,2], Joon Young Seo[1], and Kevin Yeo[1]

[1] Google LLC, {sarvar,jyseo,kwlyeo}@google.com
[2] Università di Salerno, giuper@gmail.com

Abstract. Encrypted multi-maps enable outsourcing the storage of a multi-map to an untrusted server while maintaining the ability to query privately. We focus on encrypted Boolean multi-maps that support arbitrary Boolean queries over the multi-map. Kamara and Moataz [Eurocrypt'17] presented the first encrypted multi-map, BIEX, that supports CNF queries with optimal communication, worst-case sublinear search time and non-trivial leakage.

We improve on previous work by presenting a new construction CNFFilter for CNF queries with significantly less leakage than BIEX, while maintaining both optimal communication and worst-case sublinear search time. As a direct consequence our construction shows additional resistance to leakage-abuse attacks in comparison to prior works. For most CNF queries, CNFFilter avoids leaking the result sets for any singleton queries for labels appearing in the CNF query. As an example, for the CNF query of the form $(\ell_1 \vee \ell_2) \wedge \ell_3$, our scheme does not leak the result sizes of queries to ℓ_1, ℓ_2 or ℓ_3 individually. On the other hand, BIEX does leak some of this information. This is just an example of the reduced leakage obtained by CNFFilter. The core of CNFFilter is a new *filtering* algorithm that performs set intersections with significantly less leakage compared to prior works.

We implement CNFFilter and show that CNFFilter achieves faster search times and similar communication overhead compared to BIEX at the cost of a small increase in server storage.

1 Introduction

In this work, we study the notion of *structured encryption* that was introduced by Chase and Kamara [17]. Structured encryption (STE) is a general cryptographic primitive that considers the scenario where a data owner (commonly referred to as the client) wishes to store a data structure on a potentially untrusted server such as a cloud storage provider. STE schemes should ensure that clients are able to perform all necessary data structure operations correctly over the server-stored encrypted data while ensuring that the adversarial server learns as little information as possible.

The privacy goal of a STE scheme is to reveal little information about the contents of the outsourced data structure as well as the operations that are

performed on the data structure. In an ideal world, these schemes would leak no information about either the underlying data or the executed algorithms. However, the only known ways to achieve this desired privacy is through the use of extremely expensive cryptographic primitives such as oblivious RAM [24] and/or fully homomorphic encryption [23]. In contrast, structured encryption considers a more relaxed privacy requirement with the hope of achieving the small overhead necessary for practical applications. In more detail, structured encryption are defined by a leakage function that is an upper bound on the information that may be learned by the adversarial server. As a result, some schemes have larger than ideal leakage, but avoid using heavy cryptographic primitives. However, we note that caution is necessary when picking leakage functions as there have been many works (see [27,13,33,43,35,26,25,8] as some examples) showing that various leakage profiles may be utilized by intelligent adversaries to compromise privacy in certain settings.

In our work, we will focus on an important type of STE scheme called *encrypted multi-maps*. An encrypted multi-map EMM structurally encrypts a multi-map MM consisting of pairs $(\ell, \vec{v})$ of labels ℓ and tuples $\vec{v}$ of values. We use the writing $\mathsf{MM}[\ell]$ to denote the tuple associated to label ℓ by MM. We focus on encrypted multi-maps due to its significance in a wide range of important real-world applications. Encrypted multi-maps have been the basis of many *searchable encryption* (or *encrypted search*) constructions. Searchable encryption was introduced by Song *et al.* [41] and enables a client to perform keyword searches over a corpus of documents outsourced to a server. There has been a long line of work (see [9,18,7] and references therein as examples) that considers single keyword search to determine a list of documents containing a single queried keyword. There have been many subsequent works for improving efficiency [14,21], dynamicity [42,32,36], forward and/or backward privacy [10,12],suppressing leakage [31,30,39] and improving locality [16,20,5,19] to list some examples. Faber *et al.* [22] build on [15] to obtain more complex queries such as range, substring, and wildcard queries. There also has been work for efficiency lower bounds for searchable encryption/structured encryption [38,11]. We note that many of the above searchable encryption schemes are also encrypted multi-map schemes. Chase and Kamara [17] introduced structured encryption, which is an extension of encryption for general data structures beyond search indices. Encrypted multi-maps are also used in encrypted relational databases where clients wish to perform SQL queries over encrypted databases [29].

Many previous works consider the simplest setting of encrypted multi-map schemes that enable clients to perform *exact* queries for a label ℓ and return the associated value tuple $\mathsf{MM}[\ell]$ if it exists. More recently, there has been work on improving the utility of encrypted multi-map schemes by supporting more complex and expressive queries. In our work, we focus on *encrypted Boolean multi-maps* where the client queries a Boolean formula Φ over labels and the result should be the set of values satisfying the formula Φ. For example, a query for formula $\Phi = \ell_1 \wedge \ell_2$ asks for all values v such that $(v \in \mathsf{MM}[\ell_1]) \wedge (v \in \mathsf{MM}[\ell_2])$.

This problem has been studied in several works such as [15,37]. Kamara and Moataz [28] presented BIEX[3], the first non-interactive, encrypted Boolean multi-map scheme with non-trivial leakage, optimal communication and worst-case sublinear search time. In our work, we will present new constructions with strictly smaller leakage and similar or better efficiency than all prior works.

1.1 Background and Goals

In this section, we present naive solutions and identify their shortcomings that we address in our work. Before we begin, we denote the notion of *volume* as the number of results that are associated with a specific query. We will also denote this as the *query volume*. We will also utilize the notions of response-revealing and response-hiding. Response-revealing encrypted multi-maps mean that the responses to queries are revealed to the server in plaintext. Response-hiding schemes ensure that the servers see responses in encrypted form and may only infer the size of the response.

Naive Solutions. A simple construction to enable Boolean queries is to utilize any response-hiding encrypted multi-map that can handle exact queries. For any Boolean formula Φ over labels $\ell_1, \ldots, \ell_q$, the client issues q queries, one for each of $\ell_1, \ldots, \ell_q$. The client receives all q result sets $\mathsf{MM}[\ell_1], \ldots, \mathsf{MM}[\ell_q]$ and evaluates Φ locally. The scheme is sub-optimal in terms of communication. For example, if Φ is a conjunction, the size of the result set $|\mathsf{MM}[\ell_1] \cap \ldots \cap \mathsf{MM}[\ell_q]|$ will be much smaller than the size of all q result sets, $|\mathsf{MM}[\ell_1]| + \ldots + |\mathsf{MM}[\ell_q]|$. Therefore, the server's response is larger than necessary.

To obtain optimal communication, we consider another simple encrypted Boolean multi-map that utilizes a response-revealing encrypted multi-map supporting exact queries. The client once again issues q queries, one for each of the q labels $\ell_1, \ldots, \ell_q$. The server learns the responses of all q queries and applies Φ before returning the result set to the client. The above construction obtains optimal communication as the server response consists of exactly the result set. Unfortunately, the leakage is horrible as the server learns all sets $\mathsf{MM}[\ell_1], \ldots, \mathsf{MM}[\ell_q]$ in plaintext.

The above solution can be extended to hide the plaintext values in a standard manner. Each value appearing in the multi-map will be stored as a tag (a PRF evaluation) as well as an encryption under private keys held by the client. All tags are computed under the same private key. The client issues q queries for $\ell_1, \ldots, \ell_q$ and the server learns the tags and encryptions of all values in $\mathsf{MM}[\ell_1], \ldots, \mathsf{MM}[\ell_q]$. The tags suffice to perform arbitrary set operations to apply the Boolean formula Φ since they are computed under the same key. Communication remains optimal since the server will only return encryptions of values that satisfy Φ. While the server does not learn the plaintext values in each of $\mathsf{MM}[\ell_1], \ldots, \mathsf{MM}[\ell_q]$, the server learns the volumes of the singleton queries for each of the labels $\ell_1, \ldots, \ell_q$. Using the tags, the server can perform arbitrary set

[3] While BIEX considers Boolean searchable encryption, the basic construction is an encrypted Boolean multi-map.

intersections and unions over the q results $\mathsf{MM}[\ell_1], \ldots, \mathsf{MM}[\ell_q]$ and not just the ones needed for Φ. As a consequence, the server may also learn the volumes (i.e., result sizes) for any arbitrary Boolean queries over the q labels, $\ell_1, \ldots, \ell_q$. Going forward, we will refer to this last solution as the *canonical naive solution*.

Leakage. Given the above naive solutions, an important privacy goal of encrypted Boolean multi-maps is to reduce the volume leakage for arbitrary Boolean queries. Note that optimal communication schemes must leak the volume of the query $\Phi(\ell_1, \ldots, \ell_q)$, and the goal is to limit the leakage of any additional volumes for queries that are related to the original query. Mapping this back to the canonical naive solution, we note that, since the volumes of all singleton queries for labels $\ell_1, \ldots, \ell_q$ are revealed, an adversary can compute the volume for queries $\Psi(\ell_1, \ldots, \ell_q)$ for any Boolean formula Ψ. For convenience, we define the *base query set of leakage* for the canonical naive construction as $\mathsf{B} = \{\ell_1, \ldots, \ell_q\}$ and call the span $\mathsf{Span}(\mathsf{B})$, the set of all queries for which an adversary can construct the volume from the volumes of the queries in the set B. We will formally define the notion of the base query set of leakage later. In our work, we improve the state-of-the-art by presenting non-interactive and efficient schemes with the smallest volume leakage to our knowledge.

Beyond volume leakage, we note that many encrypted multi-map schemes have non-trivial leakage about queries themselves. This leakage could include whether two Boolean queries are identical, whether a label appears in two different Boolean queries, the structure of the Boolean query, etc. For simplicity, we split off our analysis into the leakage of query volumes and all other leakage unrelated to query volumes. In our work, we ensure that our constructions have the same query leakage as prior works [15,28].

Efficiency Goals. Finally, we discuss our efficiency goals. We will aim for our constructions to be non-interactive with optimal communication and worst-case sublinear search times, while only incurring small additional storage overhead compared to prior works. To obtain optimal communication, the response of the server should be exactly the size of the query's result and the client's request size should be independent of the server-stored multi-map. Worst-case sublinear search time implies that the scheme should not unnecessarily process the entire encrypted multi-map when answering queries. Finally, the storage overhead should be small enough for practical usage.

1.2 Related Works

We survey existing constructions of encrypted Boolean multi-maps with smaller leakage compared to the canonical naive solution discussed in Section 1.1.

OXT. Cash *et al.* [15] present the oblivious cross-tag (OXT) protocol that is a non-interactive encrypted Boolean multi-map. OXT is able to handle all conjunctive queries and Boolean queries in *Searchable Normal Form* (that is, of the form $\ell_1 \wedge \Phi(\ell_2, \ldots, \ell_q)$) with worst-case sublinear search times. Unfortunately, queries for many Boolean formulae may end up having linear search times. We

note that Faber *et al.* [22] extend OXT for more (but not all) Boolean queries including range, substring and wildcard queries. The core cryptographic operation in OXT are expensive public-key operations (exponentiation in a Diffie-Hellman group). As a result, queries may end up being computationally expensive even for formulae for which the algorithm operates on a sublinear portion of the database.

BlindSeer. Pappas *et al.* [37] present BlindSeer that handles all arbitrary Boolean queries with worst-case sublinear search time unlike OXT. BlindSeer encodes the underlying multi-map using a search tree combined with Bloom filters. To traverse the tree during query time, BlindSeer utilizes secure computation to determine the next node in the search tree to visit. By using secure computation, the majority of the core cryptographic operations in BlindSeer end up being symmetric key operations. However, the search algorithm in BlindSeer still ends up being slower than OXT as the secure computation techniques require multiple rounds of client-server interactions (even if the majority of operations are symmetric-key). Given this knowledge, it is clear that reducing interaction is necessary for constructing efficient query algorithms.

BIEX. Kamara and Moataz [28] present BIEX that combines several good properties of both OXT and BlindSeer. In particular, BIEX is the first non-interactive encrypted Boolean multi-map that is able to handle arbitrary Boolean queries with worst-case sublinear search times and non-trivial leakage smaller than the canonical naive solution. Furthermore, the search algorithms of BIEX utilize only symmetric-key primitives. As a result, the search algorithm of BIEX is more efficient than both OXT and BlindSeer.

For the leakage of BIEX, consider a CNF query $\Phi = D_1 \wedge \ldots \wedge D_m$ where each clause D_i is a disjunction $(\ell_{i,1} \vee \ldots \vee \ell_{i,q_i})$. The *base query set of leakage* (see discussion in Section 1.3) consists of all singleton labels appearing in the first clause and all 2-conjunctions of labels with the first appearing in the first clause and the second label appearing in the second clause onward:

$$\{\ell_{1,i} \mid i \in [q_1]\} \cup \{(\ell_{1,i} \wedge \ell_{j,k}) \mid i \in [q_1], 2 \leq j \leq m, k \in [q_j]\}.$$

While this is significantly smaller leakage than the canonical naive solution, it includes all q_1 singleton labels appearing in the first clause. In other words, the leakage from querying Φ is at least as large as performing q_1 exact queries for all labels in the first clause, which is not ideal.

In our work, we present the first constructions with *no singleton labels in the base query set of leakage* for all Boolean queries except for disjunctions (i.e., single-term CNF). Our constructions enjoy all the good properties of BIEX including non-interaction, optimal response size, sublinear search time, and exclusive use of symmetric-key primitives.

Relation to Leakage-Abuse Attacks. Finally, we discuss prior works on leakage-abuse attacks on encrypted multi-maps (or encrypted search). The SPAR final report [4] describes data sets and query distributions that arise from real life applications. Most prior works mainly consider either exact [27,13,8] or range [33,35,26,25] queries. Given the lack of attacks, it may seem that reducing

leakage for Boolean queries is not important at first. However, it turns out that the attacks for exact or range queries may also be applied in the Boolean query setting.

Consider the leakage of BIEX as an example. For a CNF query, the adversary learns the volumes of each of q_1 labels in the first clause. In other words, the adversary could simulate and obtain leakage of q_1 exact queries using a single Boolean query to BIEX. This means that exact query attacks may be applied to encrypted Boolean multi-maps using fewer Boolean queries if we do not reduce leakage. The same idea may be applied for range query attacks. Suppose that labels come from an ordered set (such as the integers). i A single Boolean query to BIEX with q_1 labels means leakage for q_1 ranges of length 1. In the worst case when all q_1 labels are consecutive in the ordered set (such as $\{1, \ldots, q_1\}$), a single Boolean query to BIEX would end up leaking the volumes of $O(q_1^2)$ ranges ($[i, j]$ where $1 \leq i \leq j \leq q_1$).

With the above in mind, an important goal is to design encrypted Boolean multi-maps that reduce leakage. By reducing leakage, we improve the chance of our constructions resisting leakage-abuse attacks (both ones that are currently known and ones that will be developed in the future). In this work, we present constructions that avoid leaking volumes corresponding to either exact or range queries for most Boolean queries. Therefore, our construction shows additional resistance to leakage-abuse attacks in comparison to prior works.

1.3 Our Contributions

In our work, we present new encrypted Boolean multi-maps with reduced leakage and similar or better efficiency compared to prior works. As our main technical tool, we present a new *filtering* algorithm that uses only private-key primitives and performs set intersections with small leakage. By utilizing this filtering algorithm, we obtain new constructions for handling conjunctions and CNF queries with reduced leakage and optimal communication complexity as the response to a query contains exactly one ciphertext per each item in the response set. In addition, our constructions are non-interactive and require sub-linear work.

To compare leakage, we will utilize the notion of a *base query set of leakage*. Let B be the base query set of leakage for any construction. Then, the adversary may recover volumes for any Boolean query Ψ in the *span* $\mathsf{Span}(\mathsf{B})$ of the base query set of leakage B; that is, $\mathsf{Span}(\mathsf{B})$ consists of all formulae $\Psi(x_1, \ldots, x_t)$ with $x_i \in \mathsf{B}$, for $i = 1, \ldots, t$.

The worst leakage is obtained when B contains all singleton labels $\mathsf{B} = \{\ell_1, \ldots, \ell_q\}$ in which case $\mathsf{Span}(\mathsf{B})$ includes all Boolean formulae over the labels $\ell_1, \ldots, \ell_q$. This is the leakage obtained by the canonical naive solution.

Throughout our work, we will only consider constructions that satisfy all the good properties of BIEX. Our constructions will be non-interactive, handle arbitrary Boolean queries with worst-case sublinear computation and only utilize symmetric key primitives.

Our first construction $\mathsf{ConjFilter}$ supports conjunctive queries $\ell_1 \wedge \ldots \wedge \ell_q$. We will use $\mathsf{ConjFilter}$ as a building block when constructing an encrypted multi-map

for CNF queries. Even though it is only a building block, ConjFilter has smaller leakage compared to all previous efficient solutions that support conjunctions. Specifically, the base query set of leakage for ConjFilter is

$$\{(\ell_1 \wedge \ell_2), (\ell_1 \wedge \ell_2 \wedge \ell_3), (\ell_1 \wedge \ell_2 \wedge \ell_4), \ldots, (\ell_1 \wedge \ell_2 \wedge \ell_q)\}.$$

Theorem 1 (Informal). ConjFilter *is a non-interactive scheme supporting conjunctive queries that is adaptively-secure with optimal communication and worst-case sub-linear search time. For a conjunctive query* $\Phi = \ell_1 \wedge \ldots \wedge \ell_q$*, the adversary may recover the volumes of all queries of the form* $\Psi(x_1, \ldots, x_t)$ *where each* $x_i \in \{(\ell_1 \wedge \ell_2), (\ell_1 \wedge \ell_2 \wedge \ell_3), \ldots, (\ell_1 \wedge \ell_2 \wedge \ell_q)\}$ *and* Ψ *is any Boolean query.*

In particular, the queries whose volumes are leaked by ConjFilter is a subset of those leaked by prior works. As an example of the reduced leakage of ConjFilter, note that the adversary cannot recover the volume for any 2-conjunctive queries beyond $(\ell_1 \wedge \ell_2)$. In contrast, the base query set of leakage for prior non-interactive constructions OXT [15] and BIEX [28] consists of $\{(\ell_1 \wedge \ell_2), \ldots, (\ell_1 \wedge \ell_q)\}$ that enables recovering volumes for many 2-conjunctions. By playing with the base sets, one can find many queries whose volumes are leaked by prior works, but not by ConjFilter.

Next, we present an encrypted Boolean multi-map CNFFilter that supports CNF queries using ConjFilter as a building block. CNF queries are of the form $D_1 \wedge \ldots \wedge D_m$ where each clause D_i is a disjunction $(\ell_{i,1} \vee \ldots \vee \ell_{i,q_i})$ with q_i unique labels. The base query set of leakage for CNFFilter may be broken down into two parts. The first part consists of all 2-conjunctions of labels from the first and second clause:

$$\mathsf{B}' = \{(\ell_{1,i} \wedge \ell_{2,j}) \mid i \in [q_1], j \in [q_2]\}.$$

The second part consists of all 3-conjunctions of labels from the first clause, second clause and the last label appearing in the third clause onward:

$$\mathsf{B}'' = \{(b' \wedge \ell_{k,l}) \mid b' \in \mathsf{B}', 3 \leq k \leq m, l \in [q_k]\}.$$

The base query set of leakage B of CNFFilter is equal to $\mathsf{B} = \mathsf{B}' \cup \mathsf{B}''$.

Theorem 2 (Informal). CNFFilter *is a non-interactive scheme supporting CNF queries that is adaptively-secure with optimal communication and worst-case sub-linear search time. For a CNF query* $\Phi = D_1 \wedge \ldots \wedge D_m$ *where each clause* D_i *is a disjunction* $(\ell_{i,1} \vee \ldots \vee \ell_{i,q_i})$*, the adversary may recover the volumes of all queries of the form* $\Psi(x_1, \ldots, x_t)$ *where each* $x_i \in \mathsf{B}' \cup \mathsf{B}''$ *and* Ψ *is a Boolean query.*

For comparison, note that the base query set of leakage for BIEX consists of all singleton labels appearing in the first clause and all 2-conjunctions of labels with the first appearing in the first clause and the second label appearing in the second clause onward:

$$\{\ell_{1,i} \mid i \in [q_1]\} \cup \{(\ell_{1,i} \wedge \ell_{j,k}) \mid i \in [q_1], 2 \leq j \leq m, k \in [q_j]\}.$$

Note that for any CNF query Φ, the span of the base query set of leakage of CNFFilter is a subset of the one of BIEX. As a simple example of reduced leakage, unless the query is a disjunction, CNFFilter does not leak volumes for any singleton queries unlike BIEX. Many more examples of queries whose volumes are leaked by BIEX and not by CNFFilter may be found.

The above comparison only considered leakage resulting from one conjunctive/CNF query. In practice, these constructions will answer and leak information about multiple conjunctive/CNF queries. Consider the example of two queries resulting in the base query sets of leakage B_1 and B_2. In the worst case, the adversary may recover volumes of any queries of the form $\Phi(x_1, \ldots, x_t)$, where $x_i \in \mathsf{B}_1 \cup \mathsf{B}_2$. In other words, leakage may explode as more queries are performed. Therefore, it is integral to minimize the base query set leakage for individual queries.

Referring back to leakage-abuse attacks, CNFFilter does not leak volumes about exact queries except when querying disjunctions. Furthermore, the base query set of leakage for CNFFilter consists of only intersections ignoring disjunction queries. So, there is no leakage about range queries either as ranges correspond to unions of one or more consecutive labels. As a result, CNFFilter seems to be more resistant to known leakage-abuse attacks compared to BIEX.

Finally, we present a comparison of efficiency with our solution and BIEX. We obtain all the same properties including non-interaction, sublinear search times and only using symmetric-key primitives. From our implementation, we show that CNFFilter obtains faster search times than BIEX. For storage, CNFFilter only incurs 20% additional storage overhead compared to BIEX in exchange for reduced leakage and faster search times.

Both ConjFilter and CNFFilter are proved adaptively secure in the ROM. Note that the assumption of random oracles as well as their programmability are required for adaptive security by previous works [15,28] as well. Non-adaptive security for both constructions can instead be proved in the standard model.

1.4 Our Techniques

We present our new techniques used to construct ConjFilter and CNFFilter. The core of our new technique is an improved filtering algorithm for conjunctions.

Conjunctions. We start by presenting the approach to handling conjunctive queries used in previous works [15,28][4]. Consider the conjunctive query $\ell_1 \wedge \ldots \wedge \ell_q$. The main idea of prior works is to decompose the query into $(q-1)$ 2-conjunctions: $(\ell_1 \wedge \ell_2) \wedge (\ell_1 \wedge \ell_3) \wedge \ldots \wedge (\ell_1 \wedge \ell_q)$. Each of the $(q-1)$ 2-conjunction queries are computed independently such that the resulting response sets are all PRF evaluations under the key solely depending on label ℓ_1. Then, the server returns the intersection of all $q-1$ sets. In this way, the size of the server's response is proportional to the result of the query and thus optimal.

[4] In [28], the authors only present a construction for CNF queries. To derive a conjunction scheme, we consider the case where each disjunction clause is a single label.

There are several drawbacks to using this approach. The scheme leaks the volumes of the $(q-1)$ 2-conjunctions. As all response sets are PRF evaluations under the same key, the adversary may learn volumes of more complex queries. For example, the intersections of any two response sets yields the volume of a 3-conjunction. In general, the adversary can compute any Boolean function over the response sets. That is, the base query set of leakage is $\{(\ell_1 \wedge \ell_2), (\ell_1 \wedge \ell_3), \ldots, (\ell_1 \wedge \ell_q)\}$. In terms of computation cost, the server must perform computation on the order of $|\mathsf{MM}[\ell_1 \wedge \ell_2]| + \ldots + |\mathsf{MM}[\ell_1 \wedge \ell_q]|$. This is quite wasteful as the response set $\mathsf{MM}[\ell_1 \wedge \ell_2]$ is already a superset of the final response. Ideally, the server's computation should not need to be much larger than $|\mathsf{MM}[\ell_1 \wedge \ell_2]|$.

To address these drawbacks, while keeping the size of the server's response optimal, we present a new *filtering* algorithm that will be utilized by $\mathsf{ConjFilter}$. First, we compute the response set $S_2 := \mathsf{MM}[\ell_1 \wedge \ell_2]$ such that each value in S_2 is a PRF evaluation under a key depending solely on label ℓ_1. Next, we compute the intersection $S_3 := S_2 \cap \mathsf{MM}[\ell_1 \wedge \ell_3]$ by directly filtering S_2 and removing elements of S_2 that do not appear in $\mathsf{MM}[\ell_1 \wedge \ell_3]$. To do this, we maintain an additional data structure $\mathcal{X}$ that allows the server to check whether a value $v \in S_2$ belongs to $\mathsf{MM}[\ell_1 \wedge \ell_3]$ without retrieving the entire $\mathsf{MM}[\ell_1 \wedge \ell_3]$, thereby avoiding volume leakage for the query $\ell_1 \wedge \ell_3$. We repeat this filtering to compute each $S_i = S_{i-1} \cap \mathsf{MM}[\ell_1 \wedge \ell_i]$ until we compute the set S_q that is the result for the original query.

At a high level, the data structure $\mathcal{X}$ is constructed as follows. For each label pair (ℓ_a, ℓ_b) and for each value $v \in \mathsf{MM}[\ell_a \wedge \ell_b]$, $\mathcal{X}$ stores a *double tag* of v. A *double tag* of v is computed by applying two successive PRF evaluations, where the first evaluation is under the key solely depending on ℓ_a, say $K^{\mathsf{t}}_{\ell_a}$, and the second evaluation is under the key depending on ℓ_a and ℓ_b, say $K^{\mathsf{x}}_{\ell_a,\ell_b}$. Thus, given a tag of $v \in \mathsf{MM}[\ell_a \wedge \ell_b]$ under the key $K^{\mathsf{t}}_{\ell_a}$, the server can determine whether v belongs to $\mathsf{MM}[\ell_a \wedge \ell_c]$ by simply applying PRF under the key $K^{\mathsf{x}}_{\ell_a,\ell_c}$ and checking whether the resulting evaluation belongs to $\mathcal{X}$. In particular, note that the volume of $\mathsf{MM}[\ell_a \wedge \ell_c]$ is never revealed.

We note that the above filtering algorithm leaks volumes for only a subset of queries whose volumes are leaked by prior works. In particular, the base query set of leakage is $\{(\ell_1 \wedge \ell_2), (\ell_1 \wedge \ell_2 \wedge \ell_3), \ldots, (\ell_1 \wedge \ell_2 \wedge \ell_q)\}$. As an example of reduced leakage, note that the only 2-conjunction whose volume may be recovered in $\mathsf{ConjFilter}$ is $\ell_1 \wedge \ell_2$ whereas the volume of $(q-1)$ 2-conjunctions of the form $(\ell_1 \wedge \ell_2), \ldots, (\ell_1 \wedge \ell_q)$ are leaked by prior works.

We note our filtering algorithm is reminiscent but starkly different from the cross-tag protocols presented by Cash *et al.* [15]. In particular, our new filtering algorithm only use symmetric key primitives (PRFs) while the cross-tag protocols in [15] require public-key operations (i.e., exponentiation in a Diffie-Hellman group).

CNFs. Next, we show how to support CNF queries using the filtering algorithm of $\mathsf{ConjFilter}$ as a building block. We start by reviewing the BIEX construction [28] for CNF queries. Consider a CNF query of the form $D_1 \wedge \ldots \wedge D_\ell$ where $D_i = (\ell_{i,1} \vee \ldots \vee \ell_{i,q_i})$. In the first step, BIEX computes $\mathsf{MM}[D_1]$. The scheme for

computing disjunction D_1 ends up leaking the volumes for q_1 singleton queries for labels $\ell_{1,1}, \ldots, \ell_{1,q_1}$. The main problem is that there is no known scheme supporting disjunctions that do not reveal singleton query volumes.

To avoid this leakage, CNFFilter combines the first two clauses $D_1 \wedge D_2$ that may be rewritten as:

$$D_1 \wedge D_2 = (\ell_{1,1} \vee \ldots \vee \ell_{1,q_1}) \wedge (\ell_{2,1} \vee \ldots \vee \ell_{2,q_2}) = \bigvee_{i \in [q_1], j \in [q_2]} (\ell_{1,i} \wedge \ell_{2,j}).$$

In other words, $D_1 \wedge D_2$ becomes a disjunction over $q_1 q_2$ 2-conjunction queries. Next, we can apply the algorithm for computing disjunctions over the $q_1 q_2$ 2-conjunction result sets to obtain $S_2 := \mathsf{MM}[D_1 \wedge D_2]$. While the volumes of all 2-conjunction sets are revealed, no volumes for singleton queries are leaked.

We apply the filtering algorithm again to incorporate the remaining clauses $D_3, \ldots, D_m$, while keeping the server's response, and thus communication, optimal in size. If $D_3 = \ell_{3,1} \vee \ldots \vee \ell_{3,q_3}$, then $S_2 \cap \mathsf{MM}[D_3] = (S_2 \cap \mathsf{MM}[\ell_{3,1}]) \cup \ldots \cup (S_2 \cap \mathsf{MM}[\ell_{3,q_3}])$. At a high level, the filtering algorithm may be applied on S_2 for each of the labels $\ell_{3,1}, \ldots, \ell_{3,q_3}$. By repeating this for each of the clauses $D_3, \ldots, D_m$, the server successfully computes the set $\mathsf{MM}[D_1 \wedge \ldots \wedge D_m]$. The filtering scheme allows CNFFilter to avoid volume leakage for many queries.

CNFFilter also improves search times compared to BIEX. Recall that BIEX initially computes the set $\mathsf{MM}[D_1]$. Instead, CNFFilter first computes the set $\mathsf{MM}[D_1 \wedge D_2]$ that will later be filtered. As $\mathsf{MM}[D_1 \wedge D_2]$ is a subset of $\mathsf{MM}[D_1]$ and typically smaller, searching in CNFFilter ends up being faster than BIEX.

2 Preliminaries

2.1 Boolean Encrypted Multi-Maps

A multi-map data structure maintains a set of m label to value tuple pairs $\mathsf{MM} = \{(\ell_t, \vec{v}_t)\}_{t \in [m]}$, where each ℓ_i comes from the *label universe* $\mathcal{U}$ and $\vec{v}_i$ is a tuple of values where each value comes from the *value universe* $\mathcal{V}$. Different labels may be associated with tuples of different length. We assume that all m labels are unique. If any two labels are equal, then the two associated value tuples may be combined into a single value tuple.

The multi-map data structure supports the *query operation* that receives a multi-map $\mathsf{MM} = \{(\ell_t, \vec{v}_t)\}_{t \in [m]}$ and a label $\ell \in \mathcal{U}$ as arguments. If there exists $t \in [m]$ such that $\ell_t = \ell$, then the query returns $\vec{v}_t$. Otherwise, the query returns $\perp$. For convenience, if $(\ell, \vec{v}) \in \mathsf{MM}$ then we denote $\mathsf{MM}[\ell] = \vec{v}$. If ℓ does not appear in MM, then $\mathsf{MM}[\ell] = \perp$.

We consider the extended *Boolean multi-map* that enables more complex query operations beyond simply retrieving the value tuple associated with a label. More formally, a Boolean multi-map is associated with a supported class of Boolean formulae queries $\mathbb{B}$ over labels. We consider query classes: conjunctions and CNFs. For the set of conjunctions of the form $\Phi = \ell_1 \wedge \ldots \ell_q$, the query for Φ returns the intersection $\mathsf{MM}[\ell_1] \cap \ldots \cap \mathsf{MM}[\ell_q]$. For the set of CNF queries of

the form $\Phi = (\ell_{1,1} \vee \ldots \vee \ell_{1,q_1}) \wedge \ldots \wedge (\ell_{m,1} \vee \ldots \vee \ell_{m,q_m})$, the query for Φ returns the set of values $(\mathsf{MM}[\ell_{1,1}] \cup \ldots \cup \mathsf{MM}[\ell_{1,q_1}]) \cap \ldots \cap (\mathsf{MM}[\ell_{m,1}] \cup \ldots \cup \mathsf{MM}[\ell_{m,q_m}])$. For convenience, we denote the result set for any query Φ by $\mathsf{MM}[\Phi]$.

Next, we define the notion of an *encrypted Boolean multi-map*, which is the structured encryption (STE) for Boolean multi-maps. Our STE definition of encrypted Boolean multi-map will be *non-interactive*. That is, the query consists of a single client request followed by the server's reply.

Definition 1. *Let $\mathbb{B}$ be a class of Boolean formulae. A* non-interactive encrypted Boolean multi-map $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Search}, \mathsf{Resolve})$ *for the class $\mathbb{B}$ consists of the following four algorithms:*

1. $(\mathtt{msk}, \mathsf{eBMM}) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, \mathsf{MM})$*: The setup algorithm is executed by the client and takes as input the security parameter 1^λ and a multi-map* MM*. It outputs the* master secret key $\mathtt{msk}$ *and the* encrypted multi-map eBMM*. The client keeps the master secret key* $\mathtt{msk}$ *while the encrypted multi-map* eBMM *is sent to the server.*
2. $\mathtt{tok}_\Phi \leftarrow \Sigma.\mathsf{Token}(\mathtt{msk}, \Phi)$*: The token generation algorithm is executed by the client and receives the master secret key* $\mathtt{msk}$ *and a Boolean formula $\Phi \in \mathbb{B}$ as input. It returns the* token $\mathtt{tok}_\Phi$ *that is sent to the server.*
3. $\mathtt{ans} \leftarrow \Sigma.\mathsf{Search}(\mathsf{eBMM}, \mathtt{tok})$*: The search algorithm is executed by the server and takes as input the token* $\mathtt{tok}$ *sent by the client and the encrypted multi-map* eBMM*. It returns the* encrypted answer $\mathtt{ans}$ *that is sent to the client.*
4. $\mathsf{MM}[\Phi] \leftarrow \Sigma.\mathsf{Resolve}(\mathtt{msk}, \mathtt{ans})$*: The resolve algorithm is executed by the client and takes the encrypted answer* $\mathtt{ans}$ *sent by the server and the master secret key* $\mathtt{msk}$*. It computes the answer* $\mathsf{MM}[\Phi]$*.*

We impose the following natural correctness condition. For every MM *and for every $\Phi \in \mathbb{B}$, it holds that* $\Sigma.\mathsf{Resolve}(\mathtt{msk}, \mathsf{Search}(\mathsf{eBMM}, \mathtt{tok})) = \mathsf{MM}[\Phi]$*, provided that* $(\mathtt{msk}, \mathsf{eBMM}) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, \mathsf{MM})$ *and* $\mathtt{tok} = \Sigma.\mathsf{Token}(\mathtt{msk}, \Phi)$*.*

2.2 Security Notions

For encrypted Boolean multi-maps, we utilize the same security notions as typically done in structured encryption using leakage functions. The adversary's leakage is upper bounded by a pair $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$ of leakage functions. The leakage function $\mathcal{L}_{\mathsf{Setup}}$ provides an upper bound on the knowledge gained by the adversarial server when given eBMM. $\mathcal{L}_{\mathsf{Query}}$ is an upper bound on the knowledge gained by the adversary when receiving a token from the client generated using the Token algorithm and when applying the token on the encrypted multi-map in the Search algorithm.

To formalize the security notion, we use the simulation-based approach. We present definitions for adaptive adversaries. We define the following real and ideal experiments with a stateful, honest-but-curious adaptive PPT adversary $\mathcal{A}$ and a stateful, PPT simulator $\mathcal{S}$ for an encrypted Boolean multi-map $\Sigma = (\Sigma.\mathsf{Setup}, \Sigma.\mathsf{Token}, \Sigma.\mathsf{Search}, \Sigma.\mathsf{Resolve})$ for a class $\mathbb{B}$ of Boolean formulae and for leakage function $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$.

$\mathbf{Real}^{\mathsf{a}}_{\Sigma,\mathcal{A}}(1^\lambda)$:

1. The adversary $\mathcal{A}$ generates multi-map MM and passes it to the challenger $\mathcal{C}$.
2. The challenger $\mathcal{C}$ executes $(\mathtt{msk}, \mathsf{eBMM}) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, \mathsf{MM})$ and passes eBMM to the adversary $\mathcal{A}$.
3. For $i = 1, \ldots, \mathsf{poly}(\lambda)$, the adversary $\mathcal{A}$ adaptively picks Boolean formula query $\Phi_i \in \mathbb{B}$ and sends it to the challenger $\mathcal{C}$. Using Φ_i, the challenger $\mathcal{C}$ executes $\mathtt{tok}_i \leftarrow \Sigma.\mathsf{Token}(\mathtt{msk}, \Phi_i)$ and sends $\mathtt{tok}_i$ to the adversary $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

$\mathbf{Ideal}^{\mathsf{a}}_{\Sigma,\mathcal{A},\mathcal{L},\mathcal{S}}(1^\lambda)$:

1. The adversary $\mathcal{A}$ generates multi-map MM and passes it to the challenger $\mathcal{C}$.
2. The simulator $\mathcal{S}$ receives $\mathcal{L}_{\mathsf{Setup}}(\mathsf{MM})$ and returns an encrypted multi-map eBMM to the adversary $\mathcal{A}$.
3. For $i = 1, \ldots, \mathsf{poly}(\lambda)$, the adversary $\mathcal{A}$ adaptively picks Boolean formula query $\Phi_i \in \mathbb{B}$ and sends it to the challenger $\mathcal{C}$. The simulator receives $\mathcal{L}_{\mathsf{Query}}(\mathsf{MM}, \Phi_1, \ldots, \Phi_{i-1})$ from $\mathcal{C}$ and computes $\mathtt{tok}_i$ which is returned to the adversary $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

Definition 2. *The non-interactive encrypted Boolean multi-map Σ is* adaptively $\mathcal{L}$-secure *if there exists a stateful, PPT simulator $\mathcal{S}$ such that for all adaptive, PPT adversaries $\mathcal{A}$:*

$$\left|\Pr[\mathbf{Real}^{\mathsf{a}}_{\Sigma,\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}^{\mathsf{a}}_{\Sigma,\mathcal{A},\mathcal{L},\mathcal{S}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

For presentation, we split up query leakage $\mathcal{L}_{\mathsf{Query}}$ into token leakage $\mathcal{L}_{\mathsf{Token}}$ and search leakage $\mathcal{L}_{\mathsf{Search}}$. $\mathcal{L}_{\mathsf{Token}}$ encompasses all leakage derived by the adversary viewing only the search token. $\mathcal{L}_{\mathsf{Search}}$ contains all leakage from the adversary applying the search token onto the encrypted multi-map. At a high level, $\mathcal{L}_{\mathsf{Token}}$ reveals information about the query on its own such as the number of unique labels, number of CNF clauses, etc. On the other hand, $\mathcal{L}_{\mathsf{Search}}$ reveals information about the underlying multi-map. In particular, the majority of $\mathcal{L}_{\mathsf{Search}}$ consists of volume leakage for a set of queries. Suppose Q is the set of all queries whose volumes are leaked in $\mathcal{L}_{\mathsf{Search}}$. We denote the base query set of leakage S such that all queries $q \in Q$ may be written as a Boolean function $f(x_1, \ldots, x_t)$ where each $x_i \in S$. In other words, using the volumes of queries in S, one can recover the volumes for all queries in Q.

2.3 Encrypted Multi-Maps

In our work, we will utilize response-revealing encrypted multi-maps sEMM in a blackbox manner. As the name implies, response-revealing means that the values in the response are revealed to the server in plaintext. In contrast, response-hiding encrypted multi-maps ensure that the server sees each value in an encrypted manner. At a high level, response-hiding schemes will reveal only the number of values in a response as opposed to the response itself.

There are several non-interactive, adaptively-secure encrypted multi-maps with minimal leakage such as 2Lev [15] or ZMF [28] that are response-revealing. We now describe the efficiency and leakage properties of these schemes. For an MM of size n, the sEMM output by the Setup algorithm uses storage of $\Theta(n)$ ciphertexts. The Token algorithm results in a single ciphertext while the resulting answer for a keyword ℓ computed by the server using Search consists of exactly the set $\mathsf{MM}[\ell]$ in plaintext. In terms of leakage, the setup leakage of sEMM consists of the multi-set of all values that appear in the underlying multi-map; we denote this leakage $\mathcal{L}_{\mathsf{Setup}} = \mathsf{vals}(\mathsf{MM})$. As all values will be encryptions in our constructions, the setup leakage of sEMM would only consists of the number of values. The leakage during querying consists of the *query repetition pattern*, qeq, describing which two queries are performed on the same label. For a query sequence $Q = (q_1, q_2, \ldots)$, $\mathsf{qeq}(Q)$ is a $|Q| \times |Q|$ matrix M such that $M[i][j] = 1$ if and only if i-th and j-th query in Q are equal. As the scheme is response-revealing, the plaintext response, $\mathsf{resp}(\mathsf{MM}, Q) = (\mathsf{MM}[q_1], \mathsf{MM}[q_2], \ldots)$. So, $\mathcal{L}_{\mathsf{Query}} = (\mathsf{qeq}, \mathsf{resp})$.

Theorem 3. *If one-way functions exist, there exists a non-interactive, response-revealing* sEMM *that is adaptively* $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$*-secure, uses* $\Theta(n)$ *storage and* $O(|\mathsf{MM}[\ell]|)$ *ciphertexts of communication for a query to label* ℓ*.*

3 Conjunctive Queries

In this section, we present our new construction $\mathsf{ConjFilter}$ of an encrypted Boolean multi-map supporting the class of conjunctive queries. $\mathsf{ConjFilter}$ is non-interactive with optimal communication and sublinear search time. A formal description of $\mathsf{ConjFilter}$ is found in Figure 1.

3.1 Construction ConjFilter

$\mathsf{ConjFilter}$ follows BIEX [28] by pre-computing answers to all possible 2-conjunction queries, but diverges from BIEX in the method used to compute conjunctions. We start by describing the setup algorithm of $\mathsf{ConjFilter}$.

Given an input multi-map $\mathsf{MM} = \{(\ell_i, \vec{v}_i)\}_{i \in [m]}$, the setup of $\mathsf{ConjFilter}$ constructs a multi-map MM^{p} in the following way. The multi-map MM^{p} will store a tuple of values for each pair of labels (a, b) that appear in MM. The values in MM^{p} will consists of pairs of an *encrypted tag* and an *encrypted value*. For each $v \in \mathsf{MM}[\mathsf{a}] \cap \mathsf{MM}[\mathsf{b}]$, the tuple $\mathsf{MM}^{\mathsf{p}}[(\mathsf{a}, \mathsf{b})]$ will store a pair of encrypted tags and encrypted value for v. The tag for v stored in $\mathsf{MM}^{\mathsf{p}}[(\mathsf{a}, \mathsf{b})]$ is computed as $\mathsf{tag}_{\mathsf{a},v} = F(K^{\mathsf{t}}_{\mathsf{a}}, v)$ where F is a PRF. Both the tag and value are encrypted using an IND-CPA encryption scheme Enc to obtain the pair $(\mathsf{etag}_{\mathsf{a},\mathsf{b},v}, \mathsf{ev}_v) := (\mathsf{Enc}(K^{\mathsf{enc}}_{\mathsf{a},\mathsf{b}}, \mathsf{tag}_{\mathsf{a},v}), \mathsf{Enc}(K^{\mathsf{enc}}, v))$ that will be added to tuple $\mathsf{MM}^{\mathsf{p}}[(\mathsf{a}, \mathsf{b})]$. Note that the seed $K^{\mathsf{t}}_{\mathsf{a}}$ used to compute the tag depends solely on the first label of the pair a, whereas the encryption key $K^{\mathsf{enc}}_{\mathsf{a},\mathsf{b}}$ depends on both labels. K^{enc} is a system-wide encryption key. Both $K^{\mathsf{t}}_{\mathsf{a}}$ and $K^{\mathsf{enc}}_{\mathsf{a},\mathsf{b}}$ are pseudorandomly generated to ensure that the client storage remains small. The multi-map

- $(\mathtt{msk}, \mathsf{eBMM}) \leftarrow \mathsf{ConjFilter.Setup}(1^\lambda, \mathsf{MM} = \{(\ell_i, \vec{v}_i)\}_{i\in[m]})$:
  1. Randomly select PRF seeds $K^{\mathsf{p}}, K^{\mathsf{t}}, K^{\mathsf{x}} \leftarrow \{0,1\}^\lambda$.
  2. Randomly select encryption key $K^{\mathsf{enc}} \leftarrow \{0,1\}^\lambda$.
  3. Set $\mathsf{MM}^{\mathsf{p}} \leftarrow \{\ \}$.
  4. For all pairs $(\mathtt{a}, \mathtt{b})$ of labels appearing in MM:
     (a) Compute *tag seed* $K^{\mathsf{t}}_{\mathtt{a}} \leftarrow F(K^{\mathsf{t}}, \mathtt{a})$.
     (b) Compute *encryption key* $K^{\mathsf{enc}}_{\mathtt{a},\mathtt{b}} \leftarrow F(K^{\mathsf{p}}, \mathtt{a} \,||\, \mathtt{b})$.
     (c) Set $\mathsf{MM}^{\mathsf{p}}[(\mathtt{a}, \mathtt{b})] \leftarrow \emptyset$.
     (d) For all $v \in \mathsf{MM}[\mathtt{a}] \cap \mathsf{MM}[\mathtt{b}]$:
        i. Compute *tag* $\mathsf{tag}_{\mathtt{a},v} \leftarrow F(K^{\mathsf{t}}_{\mathtt{a}}, v)$ and *encrypted tag* $\mathsf{etag}_{\mathtt{a},\mathtt{b},v} = \mathsf{Enc}(K^{\mathsf{enc}}_{\mathtt{a},\mathtt{b}}, \mathsf{tag}_{\mathtt{a},v})$.
        ii. Compute *encrypted value* $\mathsf{ev}_v = \mathsf{Enc}(K^{\mathsf{enc}}, v)$.
        iii. Add $(\mathsf{etag}_{\mathtt{a},v}, \mathsf{ev}_v)$ to $\mathsf{MM}^{\mathsf{p}}[(\mathtt{a}, \mathtt{b})]$.
  5. Execute $(\mathtt{msk}^{\mathsf{p}}, \mathsf{EMM}^{\mathsf{p}}) \leftarrow \mathsf{sEMM.Setup}(1^\lambda, \mathsf{MM}^{\mathsf{p}})$.
  6. Initialize $\mathcal{X} = \emptyset$.
  7. For all pairs $(\mathtt{a}, \mathtt{b})$ of labels appearing in MM:
     (a) Compute double-tag seed $K^{\mathsf{x}}_{\mathtt{a},\mathtt{b}} = F(K^{\mathsf{x}}, \mathtt{a} \,||\, \mathtt{b})$.
     (b) For all $v \in \mathsf{MM}[\mathtt{a}] \cap \mathsf{MM}[\mathtt{b}]$:
        i. Compute double tag $F(K^{\mathsf{x}}_{\mathtt{a},\mathtt{b}}, \mathsf{tag}_{\mathtt{a},v})$ and add it to $\mathcal{X}$.
  8. Randomly permute $\mathcal{X}$.
  9. Return $(\mathtt{msk} = (K^{\mathsf{p}}, K^{\mathsf{x}}, K^{\mathsf{enc}}, \mathtt{msk}^{\mathsf{p}}), \mathsf{EMM} = (\mathsf{EMM}^{\mathsf{p}}, \mathcal{X}))$.
- $\mathtt{tok}_\Phi \leftarrow \mathsf{ConjFilter.Token}(\mathtt{msk} = (K^{\mathsf{p}}, K^{\mathsf{x}}, K^{\mathsf{enc}}, \mathtt{msk}^{\mathsf{p}}), \Phi = (\ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_q))$:
  1. Compute $\mathtt{tok}^{\mathsf{p}} \leftarrow \mathsf{sEMM.Token}(\mathtt{msk}^{\mathsf{p}}, (\ell_1, \ell_2))$.
  2. Compute *encryption key* $K^{\mathsf{enc}}_{\ell_1,\ell_2} \leftarrow F(K^{\mathsf{p}}, \ell_1 \,||\, \ell_2)$.
  3. For $d = 3, \ldots, q$:
     (a) Compute *double-tag seed* $K^{\mathsf{x}}_d = F(K^{\mathsf{x}}, \ell_1 \,||\, \ell_d)$.
  4. Return $\mathtt{tok}_\Phi = (\mathtt{tok}^{\mathsf{p}}, K^{\mathsf{enc}}_{\ell_1,\ell_2}, K^{\mathsf{x}}_3, \ldots, K^{\mathsf{x}}_q)$.
- $\mathtt{ans} \leftarrow \mathsf{ConjFilter.Search}(\mathtt{tok}_\Phi = (\mathtt{tok}^{\mathsf{p}}, K^{\mathsf{enc}}_{\ell_1,\ell_2}, K^{\mathsf{x}}_3, \ldots, K^{\mathsf{x}}_q), \mathsf{EMM} = (\mathsf{EMM}^{\mathsf{p}}, \mathcal{X}))$:
  1. Retrieve $\{(\mathsf{etag}_l, \mathsf{ev}_l)\}_{l\in[L]} \leftarrow \mathsf{sEMM.Search}(\mathtt{tok}^{\mathsf{p}}, \mathsf{EMM}^{\mathsf{p}})$.
  2. For $l = 1, \ldots, L$:
     (a) Compute $\mathsf{tag}_l \leftarrow \mathsf{Dec}(K^{\mathsf{enc}}_{\ell_1,\ell_2}, \mathsf{etag}_l)$.
  3. Set $\mathtt{ans} \leftarrow \emptyset$.
  4. For $l = 1, \ldots, L$:
     (a) For $d = 3, \ldots, q$:
        i. Compute *double tag* $\mathsf{dtag}_{l,d} \leftarrow F(K^{\mathsf{x}}_l, \mathsf{tag}_d)$.
     (b) If all $\mathsf{dtag}_{l,3}, \ldots, \mathsf{dtag}_{l,q} \in \mathcal{X}$, then add ev_l to $\mathtt{ans}$.
  5. Return $\mathtt{ans}$.
- $\mathtt{ans} \leftarrow \mathsf{ConjFilter.Resolve}((\mathsf{ev}_1, \ldots, \mathsf{ev}_r), \mathtt{msk} = (K^{\mathsf{p}}, K^{\mathsf{x}}, K^{\mathsf{enc}}, \mathtt{msk}^{\mathsf{p}}))$:
  1. Return $\{\mathsf{Dec}(K^{\mathsf{enc}}, \mathsf{ev}_1), \ldots, \mathsf{Dec}(K^{\mathsf{enc}}, \mathsf{ev}_r)\}$.

Fig. 1: Pseudocode for Construction ConjFilter.

MM^{p} is then encrypted using a response-revealing encrypted multi-map (see Section 2.3) to construct $\mathsf{EMM}^{\mathsf{p}}$ that is sent to the server. In addition to $\mathsf{EMM}^{\mathsf{p}}$, $\mathsf{ConjFilter}$ will also construct a set $\mathcal{X}$ of *double tags* that will also be stored by the server. For each pair of labels (a, b) and for each value $v \in \mathsf{MM}[\mathsf{a}] \cap \mathsf{MM}[\mathsf{b}]$, $\mathcal{X}$ will store the double tag $\mathsf{dtag}_{\mathsf{a},\mathsf{b},v} = F(K^{\mathsf{x}}_{\mathsf{a},\mathsf{b}}, \mathsf{tag}_{\mathsf{a},v})$, which is essentially a PRF evaluation of the tag $\mathsf{tag}_{\mathsf{a},v}$ that was stored in the encrypted multi-map $\mathsf{EMM}^{\mathsf{p}}$. The PRF seed $K^{\mathsf{x}}_{\mathsf{a},\mathsf{b}}$ will be pseudorandomly generated from a secret master key and the labels a and b. $\mathcal{X}$ is the new structure of $\mathsf{ConjFilter}$ that enables filtering in a way that reduces volume leakage.

To answer a query for conjunction $\ell_1 \wedge \ldots \wedge \ell_q$, the client issues a query token to $\mathsf{EMM}^{\mathsf{p}}$ for (ℓ_1, ℓ_2). As $\mathsf{EMM}^{\mathsf{p}}$ is response-revealing, the server will learn the entry $\mathsf{MM}^{\mathsf{p}}[(\ell_1, \ell_2)]$ in plaintext. In addition, the client also sends the encryption key $K^{\mathsf{enc}}_{\ell_1,\ell_2}$ that enables the server to decrypt all encrypted tags that appear in $\mathsf{MM}^{\mathsf{p}}[(\ell_1, \ell_2)]$. As a result, the server may decrypt the encrypted tags in $\mathsf{MM}[\ell_1 \wedge \ell_2]$ to obtain set $S_2 = \{(\mathsf{tag}_1, \mathsf{ev}_1), (\mathsf{tag}_2, \mathsf{ev}_2), \ldots\}$ of pairs of tags and encrypted values. Note, the server may only decrypt encrypted tags but may not decrypt the encrypted values. Next, we want to filter S_2 to only keep pairs of tags and encrypted values that correspond to values that appear in $\mathsf{MM}[\ell_3]$. To do this, we utilize the set of double tags $\mathcal{X}$. In the request, the client issues the PRF seed $K^{\mathsf{x}}_{\ell_1,\ell_3}$ for filtering S_2 with $\mathsf{MM}[\ell_3]$. For each tag_i in S_2, the server computes $F(K^{\mathsf{x}}_{\ell_1,\ell_3}, \mathsf{tag}_i)$ and checks whether the PRF evaluation appears in $\mathcal{X}$. The server computes the set $S_3 \subseteq S_2$ such that S_3 contains the pair $(\mathsf{tag}_i, \mathsf{ev}_i)$ from S_2 if and only if $F(K^{\mathsf{x}}_{\ell_1,\ell_3}, \mathsf{tag}_i) \in \mathcal{X}$. We note that S_3 consists only of the tag and encrypted value pairs corresponding to values that appear in $\mathsf{MM}[\ell_1 \wedge \ell_2 \wedge \ell_3]$. As a result, the server successfully filters S_2 to keep elements that also appear in $\mathsf{MM}[\ell_3]$. By repeating the filtering algorithm for each $\ell_3, \ldots, \ell_q$, the server will exactly compute an encrypted version of $\mathsf{MM}[\ell_1 \wedge \ldots \wedge \ell_q]$.

We note that handling singleton queries (1-conjunctions) of the form ℓ is a special case where only a single query to $\mathsf{EMM}^{\mathsf{p}}$ for entry ℓ is issued by the client. The server returns the response set that may be decrypted by the client to obtain $\mathsf{MM}[\ell]$. For convenience, we do not add this special case of Figure 1 to focus the pseudocode on the new techniques.

3.2 Efficiency

The encrypted multi-map of $\mathsf{ConjFilter}$ consists of two structures: $\mathsf{EMM}^{\mathsf{p}}$ and $\mathcal{X}$. Altogether, both structures store three objects for each value appearing in $\mathsf{MM}[\mathsf{a}] \cap \mathsf{MM}[\mathsf{b}]$ for each pair of labels (a, b). Therefore, the encrypted multi-map $\mathsf{ConjFilter}$ requires server storage $O(\sum_{\mathsf{a},\mathsf{b} \in \mathcal{U}} |\mathsf{MM}[\mathsf{a}] \cap \mathsf{MM}[\mathsf{b}]|)$. The client only stores $O(1)$ PRF seeds and encryption keys.

For communication, consider a conjunctive query Φ over q labels. The token $\mathtt{tok}$ for Φ contains a token for $\mathsf{EMM}^{\mathsf{p}}$, a decryption key and $q-2$ double-tag seeds. As the token size of $\mathsf{EMM}^{\mathsf{p}}$ is $O(1)$, we get that the total size of tokens for $\mathsf{ConjFilter}$ is $O(q)$. In particular, the token size is independent on the size of the underlying multi-map. The server response size is optimal as there is exactly one ciphertext returned for each value that appears in the response $\mathsf{MM}[\Phi]$.

Finally, we consider the computational cost of the server performing the query. Note, the server first computes a response set for the query $\mathsf{MM}[\ell_1 \wedge \ell_2]$. Afterwards, the server filters the set for each of the other labels $\ell_3, \ldots, \ell_q$. As a result, the server computation becomes $O(q \cdot |\mathsf{MM}[\ell_1 \wedge \ell_2]|)$. In the natural setting that $|\mathsf{MM}[\ell_1] \cap \mathsf{MM}[\ell_2]|$ is sublinear in the size of the input multi-map, $\mathsf{ConjFilter}$ performs sublinear work in the input multi-map.

3.3 Formal Description of Leakage for ConjFilter

In this section, we give a formal description of the leakage function for $\mathsf{ConjFilter}$. For our leakage descriptions, we consider an input multi-map MM and a sequence $Q = (\Phi^1, \Phi^2, \ldots)$ of conjunctive queries, where the i-th query is the conjunction $\Phi^i = (\ell_1^i \wedge \ldots \wedge \ell_{q_i}^i)$. We split the information leaked by $\mathsf{ConjFilter}$ for an input multi-map MM and a query sequence Q into three leakages:

1. The *setup leakage*, $\mathcal{L}_{\mathsf{Setup}}$, learned by the adversary from viewing the encrypted Boolean multi-map $(\mathsf{EMM}^{\mathsf{p}}, \mathcal{X})$;
2. The *token leakage*, $\mathcal{L}_{\mathsf{Token}}$, learned by the adversary from viewing the tokens;
3. The *search leakage*, $\mathcal{L}_{\mathsf{Search}}$, learned by the adversary when applying the tokens to the encrypted Boolean multi-map $(\mathsf{EMM}^{\mathsf{p}}, \mathcal{X})$.

Query leakage, $\mathcal{L}_{\mathsf{Query}} = (\mathcal{L}_{\mathsf{Token}}, \mathcal{L}_{\mathsf{Search}})$, is the union of token and search leakage.

Before presenting our leakage, we define the notion of *repetition patterns*. Note that $\mathsf{ConjFilter}$ makes extensive use of PRFs to compute various cryptographic objects. As PRF functions are deterministic, this means that these objects might repeatedly appear several times in query tokens or during server processing. In the description of the leakage of our constructions we will make use of repetition patterns to encode information about the appearances of an object. In general, suppose we have T occurrences of an object. For a fixed ordering of the T occurrences, the repetition pattern will consist of a sequence of T integers, one for each occurrence of the object. Each integer will correspond to the first index of this object was encountered. Two entries of the sequence are equal if and only if they correspond to the same instance of the object. An example of the repetition pattern is the query equality pattern appearing in many encrypted multi-map schemes (such as [31,39]) that reveals whether two queries are the same as well as the first time this query was previously seen. We will utilize repetition patterns for tags, double tags, decryption keys and PRF seeds used to compute double tags.

Setup Leakage. The setup leakage $\mathcal{L}_{\mathsf{Setup}}$ is the information learned by the adversary from the encrypted multi-map $\mathsf{EMM}^{\mathsf{p}}$ and set $\mathcal{X}$ of double tags computed by $\mathsf{ConjFilter.Setup}$. We utilize a response-revealing encrypted multi-map from Section 2.3. Recall that the setup leakage of $\mathsf{EMM}^{\mathsf{p}}$ is the set of values appearing in the underlying input multi-map MM^{p}. As all values are pairs of encryptions, the leakage is simply the size of MM^{p}. Each element of $\mathcal{X}$ is a PRF evaluation. Therefore, the adversary learns no information beyond the size of $\mathcal{X}$, that is identical to the size of MM^{p}. To complete the description, the setup leakage is

$\mathcal{L}^{\mathsf{st}}_{\mathsf{ConjFilter}}(\mathsf{MM}) = N = \sum_{\ell,\ell' \in \mathcal{U}} |\mathsf{MM}[\ell] \cap \mathsf{MM}[\ell']|$, which is the size of MM^{p} and $\mathcal{X}$.

Token Leakage. The token leakage consists of the information learned by the adversary from tokens. First of all, note that the number of double tags in the i-th token leaks the number of labels appearing in i-th query. We denote this leakage function by $\#\mathsf{labels}(Q) = (q_1, \ldots, q_{|Q|})$.

Next, we note that the encryption key in the token of the i-th query is pseudorandomly generated using the first two labels, ℓ_1^i and ℓ_2^i, of the query. Therefore if two queries share the first two labels, the corresponding tokens contain the same encryption key. Thus $\mathsf{EMM}^{\mathsf{p}}$ leaks the *encryption key repetition pattern* denoted by $\mathsf{encryptionKeyRP}$. Formally, $\mathsf{encryptionKeyRP}$ is an array whose length is the number of queries and $\mathsf{encryptionKeyRP}[i]$ is the smallest $j \leq i$ such that the i-th and the j-th tokens contain the same encryption key.

Similarly, the leakage also consists of *double tag PRF seed repetition patterns* denoted by $\mathsf{doubleTagSeedRP}$. Similarly, $\mathsf{doubleTagSeedRP}$ is an array whose length is the number of double-tag PRF seeds seen and each entry is an index of when the corresponding double-tag PRF seed was first encountered.

A similar leakage is obtained from double tags. Consider two queries Φ^i and Φ^j with the same first label such that the s-th label of Φ^i is equal to the t-th label of Φ^j. That is,

$$\ell_1^i = \ell_1^j \text{ and } \ell_s^i = \ell_t^j.$$

Then double-tag PRF seed $K^{\mathsf{x}}_{i,s}$ in the query token for Φ^i is equal to double-tag PRF seed $K^{\mathsf{x}}_{j,t}$ in the query token for Φ^j. We encode these repetitions in array $\mathsf{doubleTagSeedRP}$ where, for each (i, s), $\mathsf{doubleTagSeedRP}[i, s] = (j, t)$ where $j \leq i$ is the smallest index such that the j-th query has the same t-th label as the s-th label of the i-th query. The token leakage is thus set to $\mathcal{L}^{\mathsf{t}}_{\mathsf{ConjFilter}} = (\#\mathsf{labels}, \mathsf{encryptionKeyRP}, \mathsf{doubleTagSeedRP})$.

Search Leakage. For search leakage, we note that the server sees in the plaintext both tags and double tags. As a result, the search leakage consists of the tag and double tag repetition patterns tagRP and $\mathsf{doubleTagRP}$. The leakage tagRP is an array whose length is equal to the number of tags seen. Each entry corresponds to the index that the tag was first seen. The function $\mathsf{doubleTagRP}$ is defined similarly for double tags.

When the tokens are applied to the Boolean encrypted multi-map, the adversary sees tags (obtained by decrypting the encrypted tags from MM^{p}) and the double tags and which of them belongs to $\mathcal{X}$. Thus the execution of search leaks the number and the repetition pattern tagRP of tags, and the number, the repetition pattern $\mathsf{doubleTagRP}$ and the membership in $\mathcal{X}$ of the double tags.

Let us discuss what this tells us about MM and Q, starting from the tags.

The number L_i of tags obtained from $\mathsf{EMM}^{\mathsf{p}}$ in the i-th search invocation corresponds to the size of $\mathsf{MM}[\ell_1^i \wedge \ell_2^i]$. To understand the tag repetition pattern, note that the tag is a function of the first label of a query and of the actual value v. Thus if the l_1-th tag of query i_1 is the same as the l_2-th tag of query i_2, the two queries have the same first label, that is $\ell_1^{i_1} = \ell_1^{i_2}$, and there exists

$v \in \mathsf{MM}[\ell_1^{i_1} \wedge \ell_1^{i_2}]$. Therefore, by counting the number of common tags between query i_1 and query i_2, it is possible to compute the size of

$$\mathsf{MM}[\ell_1^{i_1} \wedge \ell_2^{i_1} \wedge \ell_1^{i_2} \wedge \ell_2^{i_2}] = \mathsf{MM}[\ell_1^{i_1} \wedge \ell_2^{i_1} \wedge \ell_2^{i_2}].$$

This can be extended to compute the size of conjunction of four or more labels that come from queries with the same first label.

For the double tags, observe that a double tag is obtained from a tag and a double-tag seed and thus associated repetition pattern $\mathsf{doubleTagRP}$ can be obtained from the tag repetition pattern tagRP and the double-tag seed repetition pattern $\mathsf{doubleTagSeedRP}$. We include it in the leakage for convenience. Membership in $\mathcal{X}$ of double tags can be encoded by q matrices $\mathsf{MX}_1, \ldots, \mathsf{MX}_q$, one for each query, defined as follows:

$$\mathsf{MX}_i[l][d] = \begin{cases} 1, & \text{if } \mathsf{dtag}_{l,d} \in \mathcal{X} \text{ for the } i\text{-th query;} \\ 0, & \text{otherwise.} \end{cases}$$

By counting the number of 1's in column d of MX_i, one obtains the size of $\mathsf{MM}[\ell_1^i \wedge \ell_2^i \wedge \ell_d^i]$. This can be extended to the computation of the size of conjunctions of four or more labels, by counting the number of common rows that contain 1 in two or more columns.

Finally, we note that the server learns whether a double tag appears in the set $\mathcal{X}$ or not. For each query Φ^i, we denote the leakage MX_i as an array of length $|\mathsf{MM}^{\mathsf{p}}[(\ell_1^i, \ell_i^2)]| \cdot (q-2)$ with one entry for each double tag seen when processing Φ^i. An entry of MX_i is 1 if and only if the corresponding double tag appears in $\mathcal{X}$ or not. Recall that a double tag is pseudorandomly generated based on two labels (a, b) and a value v. If the corresponding MX_i entry is 1, it means that the value appears in the intersection of $\mathsf{MM}[\mathsf{a}] \cap \mathsf{MM}[\mathsf{b}]$. Therefore, we have that $\mathcal{L}_{\mathsf{Search}} = (\mathsf{tagRP}, \mathsf{doubleTagRP}, \{\mathsf{MX}_i\}_{i \in [|Q|]})$.

We can re-interpret the volume leakage of $\mathcal{L}_{\mathsf{Search}}$ to determine the base query set of leakage with respect to a single conjunctive query $\ell_1 \wedge \ldots \wedge \ell_q$. Note the query to $\mathsf{MM}[\ell_1 \wedge \ell_2]$ reveals the volume of $(\ell_1 \wedge \ell_2)$. The double tags reveal the volumes of $(\ell_1 \wedge \ell_2 \wedge \ell_i)$ for all $i \geq 3$. As all the sets of PRF evaluations are under the same key, the adversary may perform arbitrary set intersections and unions over the responses. Therefore, the adversary learns the volume of any query of the form $\Psi(x_1, \ldots, x_t)$ where $x_i \in \mathsf{B} = \{(\ell_1 \wedge \ell_2), (\ell_1 \wedge \ell_2 \wedge \ell_3), \ldots, (\ell_1 \wedge \ell_2 \wedge \ell_q)\}$ where B is the base query set of leakage. The above analysis works when the query is a conjunction of two or more labels. For singleton label queries, the volume of the single queried label is leaked, which is unavoidable when insisting on optimal download sizes.

The query leakage consists of both the token and search leakage, $\mathcal{L}_{\mathsf{Query}} = (\mathcal{L}_{\mathsf{Token}}, \mathcal{L}_{\mathsf{Search}})$. We prove the following theorem in the full version.

Theorem 4. ConjFilter *is an adaptively* $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$*-secure encrypted Boolean multi-map scheme that supports conjunctive queries in the random oracle model.*

3.4 Comparison with BIEX [28]

For completeness, we present a comprehensive overview of the techniques used in BIEX in our full version. These similar ideas were also used in prior works such as [15]. In terms of setup and token leakage, it turns out that both ConjFilter and BIEX have identical leakage. The main difference in leakage occurs during search time. To exhibit the differences, we start by comparing the set of plaintext tags that are revealed to the server. For a query $\Phi = \ell_1 \wedge \ldots \ell_q$, ConjFilter only reveals the tags appearing in the multi-map entry $\mathsf{MM}^{\mathsf{p}}[\ell_1 \wedge \ell_2]$. On the other hand, BIEX reveals all plaintext tags appearing in $q-1$ multi-map entries $\mathsf{MM}^{\mathsf{p}}[\ell_1 \wedge \ell_2], \ldots, \mathsf{MM}^{\mathsf{p}}[\ell_1 \wedge \ell_q]$. As an immediate consequence, ConjFilter only leaks volumes for a single 2-conjunction $(\ell_1 \wedge \ell_2)$ while BIEX leaks volumes for $q-1$ 2-conjunctions $(\ell_1 \wedge \ell_2), \ldots, (\ell_1 \wedge \ell_q)$.

Note that ConjFilter reveals double tags that do not exist in BIEX. The leakage reveals whether the double tag corresponding to labels (ℓ_1, ℓ_i) and a value $v \in \mathsf{MM}[\ell_1 \wedge \ell_2]$ appears in $\mathcal{X}$. Note this is true if and only if $v \in \mathsf{MM}[\ell_1 \wedge \ell_2 \wedge \ell_i]$. Therefore, ConjFilter ends up leaking the volumes of 3-conjunctions of the form $\ell_1 \wedge \ell_2 \wedge \ell_i$ where $i \in \{3, \ldots, q\}$. These are the only sets of PRF evaluations that are leaked by ConjFilter on top of the 2-conjunction result $\ell_1 \wedge \ell_2$. As these sets are evaluated under the same PRF key, the adversary may perform arbitrary set operations over them to derive volumes of other queries. Therefore, the base query set of leakage is $\{(\ell_1 \wedge \ell_2), (\ell_1 \wedge \ell_2 \wedge \ell_3), \ldots, (\ell_1 \wedge \ell_2 \wedge \ell_q)\}$.

Going back to BIEX, the only sets of PRF evaluations leaked consist of 2-conjunctions from the set $\{(\ell_1 \wedge \ell_2), \ldots, (\ell_1 \wedge \ell_q)\}$. This ends up being the base set of query leakage as all PRF evaluations are under the same key. It is easy to see that the span of the base query set of leakage of ConjFilter is a subset of the span of the base query set of leakage of BIEX. This means that BIEX ends up leaking volumes of more queries. To see some concrete reduced leakage, BIEX already leaks volumes of more 2-conjunctions than ConjFilter. Looking at 3-conjunctions, ConjFilter leaks only 3-conjunctions of the form $(\ell_1 \wedge \ell_2 \wedge \ell_3), \ldots, (\ell_1 \wedge \ell_2 \wedge \ell_q)$. On the other hand, BIEX leaks volumes for 3-conjunctions of the form $\ell_1 \wedge \ell_i \wedge \ell_j$ where $i < j \in \{2, \ldots, q\}$. Therefore, it is clear ConjFilter leaks volumes for less 3-conjunctions than BIEX. One can find many more queries for which volumes are leaked by BIEX and not ConjFilter using the base sets. As leakage explodes as more conjunctive queries are handled by BIEX and ConjFilter, the leakage reduction on ConjFilter only gets better when considering leakage of multiple conjunctive queries.

As a caveat, we note that ConjFilter and BIEX leak volumes for identical query sets in only two cases. The first case is singleton label queries where it is necessary to leak the response size due to optimal communication requirements. The other case is 2-conjunctions where both schemes leak only the volume of the 2-conjunctive query. For conjunctions with 3 or more labels, the set of queries for which volumes are leaked for ConjFilter is always a strict subset of BIEX.

4 CNF Queries

In this section, we present CNFFilter, a construction supporting general CNF queries that extends the filtering techniques of ConjFilter. The formal description of CNFFilter may be found in Figure 2.

4.1 Construction CNFFilter

The CNFFilter.Setup algorithm is identical to ConjFilter.Setup that computes the encrypted multi-map $\mathsf{EMM}^{\mathsf{p}}$ and set $\mathcal{X}$.

Next, we show how CNFFilter handles CNF queries using $\mathsf{EMM}^{\mathsf{p}}$ and $\mathcal{X}$. We start with the simple case of a CNF formulae $\Phi = D_1 \wedge D_2$ with only two clauses where each clause $D_d = \ell_{d,1} \vee \ldots \vee \ell_{d,q_d}$, for $d = 1, 2$. For all $i \in \{1, \ldots, q_1\}$, we define the set S_i as

$$S_i := (\mathsf{MM}[\ell_{1,i}] \cap \mathsf{MM}[D_2]) \setminus \left(\mathsf{MM}[\ell_{1,i}] \cap \mathsf{MM}[D_2] \cap \left(\bigcup_{r=i+1}^{q_1} \mathsf{MM}[\ell_{1,r}] \right) \right).$$

Note that any two sets, S_i and S_j, are disjoint as long as $i \neq j$. Furthermore, the union of all q_1 sets is exactly $\mathsf{MM}[D_1 \wedge D_2]$. In other words, $S_1, \ldots, S_{q_1}$ is a partition of $\mathsf{MM}[D_1 \wedge D_2]$ and this is crucial to obtain optimal communication. Let us show how the search algorithm will compute the sets $S_1, \ldots, S_{q_1}$. Its output will consists of the union of the q_1 sets.

The client will issue tokens $\mathtt{tok}_1, \ldots, \mathtt{tok}_{q_1}$ to compute each of the sets $S_1, \ldots, S_{q_1}$. The first part of $\mathtt{tok}_i$ corresponding to S_i will be the q_2 tokens to query entries $(\ell_{1,i}, \ell_{2,j})$, for all $j \in \{1, \ldots, q_2\}$, in $\mathsf{EMM}^{\mathsf{p}}$. Additionally, $\mathtt{tok}_i$ will contain the encryption keys to decrypt all tags that appear in the tuples $\mathsf{MM}^{\mathsf{p}}[(\ell_{1,i}, \ell_{2,j})]$, for all $j \in \{1, \ldots, q_2\}$. As a result, the server will be able to obtain the tags in the tuples $\mathsf{MM}[\ell_{1,i} \wedge \ell_{2,1}], \ldots, \mathsf{MM}[\ell_{1,i} \wedge \ell_{2,q_2}]$. Using the tags, the server may also compute the union of all q_2 sets, which we denote as S_i, that is a superset of the final answer. Note, that S_i is currently equal to the set $\mathsf{MM}[\ell_{1,i}] \cap \mathsf{MM}[D_2]$. Two different parts S_i and S_j might not be disjoint at the moment. For example, there might be a value $v \in \mathsf{MM}[\ell_{1,i}] \cap \mathsf{MM}[\ell_{1,j}] \cap \mathsf{MM}[D_2]$ that appears in both S_i and S_j. To ensure all parts are disjoint, and thus guarantee optimal communication, we filter each S_i and remove all values that will appear in sets $S_{i+1}, S_{i+2}, \ldots, S_{q_1}$. If any value v appears in $S_i \cap S_j$, it must appear in $\mathsf{MM}[\ell_{1,i} \wedge \ell_{1,j}]$. Therefore, we can use $\mathcal{X}$ to filter any values in S_i that also will also appear in S_j. To do this for any $i < j$, the client sends the PRF seed $K^{\mathsf{x}}_{\ell_{1,i},\ell_{1,j}}$. The server computes the PRF evaluation of every tag in S_i using $K^{\mathsf{x}}_{\ell_{1,i},\ell_{1,j}}$ (i.e. the double tag). Every pair whose double tag appears in $\mathcal{X}$ may be safely removed from S_i as it will appear in S_j. After filtering all sets $S_1, \ldots, S_{q_1}$, the server obtains a partitioning of $\mathsf{MM}[D_1 \wedge D_2]$.

Next, we explain the extension to CNF queries with any number of clauses. As described above, we have successfully retrieved the q_1 sets $S_1, \ldots, S_{q_1}$ whose union is the answer to the query $D_1 \wedge D_2$. Given a new clause $D_3 = (\ell_{3,1} \vee \ldots \vee \ell_{3,q_3})$, we show how to compute the filtered sets $S_1 \cap \mathsf{MM}[D_3], \ldots, S_{q_1} \cap \mathsf{MM}[D_3]$

whose union corresponds to the response to the query $D_1 \wedge D_2 \wedge D_3$. Recall that all tags in each set S_i are computed using a PRF seed depending solely on label $\ell_{1,i}$. It suffices to remove all items in S_i that do not appear in any of the sets $\mathsf{MM}[\ell_{1,i} \wedge \ell_{3,1}], \ldots, \mathsf{MM}[\ell_{1,i} \wedge \ell_{3,q_3}]$. To do this, we once again use filtering via the set $\mathcal{X}$. The client will send the PRF seeds $K^{\mathsf{x}}_{\ell_{1,i},\ell_{3,1}}, \ldots, K^{\mathsf{x}}_{\ell_{1,i},\ell 3,q_3}$ and applies each of them to each tag in S_i and checks whether the resulting double tag appears in $\mathcal{X}$ or not. If any value in S_i whose corresponding q_3 double tags do not appear in $\mathcal{X}$, the value will be removed from S_i as it does not appear in $\mathsf{MM}[D_1 \wedge D_2 \wedge D_3]$. By removing all these tags, the server successfully computes $S_i \wedge D_3$ for all q_1 sets. For a CNF query of the form $D_1 \wedge D_2 \wedge \ldots \wedge D_\ell$, we can repeat the above filtering for all $D_3, \ldots, D_\ell$ to compute the final response.

We note the above description considers CNF queries with at least two clauses. For the special case of a CNF query with a single clause, the query will be a disjunction. In this case, we revert to the same algorithms for BIEX [28]. No additional storage is necessary as BIEX only requires $\mathsf{EMM}^{\mathsf{p}}$. To our knowledge, there is no way to serve disjunctions without leaking volumes of singleton labels. We leave it as an important open question to answer whether it is possible to compute disjunctions without leaking volumes of singleton labels. We omit the special case from the pseudocode in Fig. 2 to focus on our new techniques.

4.2 Efficiency

The storage of $\mathsf{CNFFilter}$ is identical to $\mathsf{ConjFilter}$ as they store the same structures $\mathsf{EMM}^{\mathsf{p}}$ and $\mathcal{X}$. So, $\mathsf{CNFFilter}$ stores $O(\sum_{\mathsf{a},\mathsf{b} \in \mathcal{U}} |\mathsf{MM}[\mathsf{a}] \cap \mathsf{MM}[\mathsf{b}]|)$ ciphertexts.

Moving on, we consider the costs of computing CNF queries of the form $\Phi = D_1 \wedge \ldots \wedge D_\ell$ where each D_i is a disjunction over q_i keys. For convenience, we denote $q = q_1 + \ldots + q_\ell$. The token for Φ contains a $\mathsf{EMM}^{\mathsf{p}}$ token and an encryption key for each pair of keys (a, b) where a appears in the first clause D_1 and b appears in the second clause D_2. As a result, there are $O(q_1 q_2)$ such keys and tokens. Additionally, for each key appearing in any of the clauses $D_3, \ldots, D_\ell$ and each key appearing in the first D_1, the token for Φ contains a PRF key. This results in an additional $O(q_1 \cdot (q - q_1 - q_2))$ PRF keys. So, the token size of $\mathsf{CNFFilter}$ is $O(q_1 \cdot q) = O(q^2)$, which is independent of the stored multi-map. The server response size is optimal as there is exactly one ciphertext returned for each value in the response $\mathsf{MM}[\Phi]$.

In terms of server computation, the server computes response sets for all queries of the form $\mathsf{MM}[\mathsf{a} \wedge \mathsf{b}]$ where a is a label from the clause D_1 and b is a label from the clause D_2. We may upper bound the size of all these $q_1 \cdot q_2$ responses by $O(q_1 \cdot q_2 \cdot |\mathsf{MM}[D_1 \wedge D_2]|)$. Each tag that appears in the response set is hashed using an additional PRF key depending on another label that appears in any of the clauses $D_3, \ldots, D_\ell$. This incurs an additional $O((q - q_1 - q_2) \cdot |\mathsf{MM}[D_1 \wedge D_2]|)$ server computation. Altogether, the total server computation is $O(q \cdot q_1 \cdot |\mathsf{MM}[D_1 \wedge D_2]|) = O(q^2 \cdot |\mathsf{MM}[D_1 \wedge D_2])$. This is sublinear in the input multi-map size as long as $\mathsf{MM}[D_1 \wedge D_2]$ is sublinear in the input multi-map size. On average, our scheme has smaller server computation as it depends only on $|\mathsf{MM}[D_1 \wedge D_2]$ compared to BIEX whose server computation depends on the

- $(\mathtt{msk}, \mathsf{EMM}) \leftarrow \mathsf{CNFFilter.Setup}(1^\lambda, \mathsf{MM} = \{(\ell_t, \vec{v}_t)\}_{t\in[m]})$:
  1. Compute $(\mathtt{msk}, \mathsf{EMM}) \leftarrow \mathsf{ConjFilter.Setup}(1^\lambda, \mathsf{MM})$.
  2. Return $(\mathtt{msk}, \mathsf{EMM})$.
- $\mathtt{tok}_\Phi \leftarrow \mathsf{CNFFilter.Token}(\mathtt{msk} = (K^{\mathsf{p}}, K^{\mathsf{x}}, K^{\mathsf{enc}}, \mathtt{msk}^{\mathsf{p}}), \Phi = D_1 \wedge \ldots \wedge D_\ell)$:
  1. For $d = 1, \ldots, \ell$, parse D_d as $(\ell_{d,1} \vee \ldots \vee \ell_{d,q_d})$.
  2. For $i = 1, \ldots, q_1$:
     (a) For $j = 1, \ldots, q_2$:
        i. Compute $K^{\mathsf{enc}}_{i,j} \leftarrow F(K, \ell_{1,i} \,||\, \ell_{2,j})$.
        ii. Compute $\mathtt{tok}^{\mathsf{p}}_{i,j} \leftarrow \mathsf{sEMM.Token}(\mathtt{msk}^{\mathsf{p}}, (\ell_{1,i}, \ell_{2,j}))$.
  3. For $i = 1, \ldots, q_1$:
     (a) For $r = i+1 \ldots, q_1$:
        i. Compute $K^{\mathsf{x}}_{\ell_{1,i},\ell_{1,r}} = F(K^{\mathsf{x}}, \ell_{1,i} \,||\, \ell_{1,r})$.
  4. For $d = 3, \ldots, \ell$:
     (a) For $i = 1, \ldots, q_1$:
        i. For $r = 1, \ldots, q_d$:
           A. Compute $K^{\mathsf{x}}_{\ell_{1,i},\ell_{d,r}} = F(K^{\mathsf{x}}, \ell_{1,i} \,||\, \ell_{d,r})$.
  5. Return $(\{(K^{\mathsf{enc}}_{i,j}, \mathtt{tok}^{\mathsf{p}}_{i,j})\}_{(i,j)\in[q_1]\times[q_2]}, \{K^{\mathsf{x}}_{\ell_{1,i},\ell_{1,r}}\}_{i<r\in[q_1]\times[q_1]}, \{K^{\mathsf{x}}_{\ell_{1,i},\ell_{3,j}}\}_{(i,j)\in[q_1]\times[q_3]}, \ldots, \{K^{\mathsf{x}}_{\ell_{1,i},\ell_{\ell,j}}\}_{(i,j)\in[q_1]\times[q_\ell]})$.
- $\mathtt{ans} \leftarrow \mathsf{CNFFilter.Search}(\mathtt{tok}_\Phi, \mathsf{EMM} = (\mathsf{EMM}^{\mathsf{p}}, \mathcal{X}))$.
  1. Parse $\mathtt{tok}_\Phi = (\{(K^{\mathsf{enc}}_{i,j}, \mathtt{tok}^{\mathsf{p}}_{i,j})\}_{(i,j)\in[q_1]\times[q_2]}, \{K^{\mathsf{x}}_{\ell_{1,i},\ell_{1,r}}\}_{i<r\in[q_1]\times[q_1]}, \{K^{\mathsf{x}}_{\ell_{1,i},\ell_{3,j}}\}_{(i,j)\in[q_1]\times[q_3]}, \ldots, \{K^{\mathsf{x}}_{\ell_{1,i},\ell_{\ell,j}}\}_{(i,j)\in[q_1]\times[q_\ell]})$.
  2. For $i = 1, \ldots, q_1$: `# Compute partition of` $\mathsf{D}_1 \wedge \mathsf{D}_2$
     (a) Set $S_i \leftarrow \emptyset$.
     (b) For $j = 1, \ldots, q_2$:
        i. Set $S_i \leftarrow S_i \cup \mathsf{sEMM.Search}(\mathtt{tok}^{\mathsf{p}}_{i,j}, \mathsf{EMM}^{\mathsf{p}})$.
     (c) Use decryption key $K^{\mathsf{enc}}_{i,j}$ to decrypt the first component of every pair of S_i and remove pairs from S_i until all pairs have distinct first component.
     (d) Parse S_i as $\{(\mathsf{tag}_1, \mathsf{ev}_1), \ldots, (\mathsf{tag}_{|S_i|}, \mathsf{ev}_{|S_i|})\}$.
     (e) For each $(\mathsf{tag}, \mathsf{ev}) \in S_i$:
        i. Compute double tag $\mathsf{dtag}_r \leftarrow F(K^{\mathsf{x}}_{\ell_{1,i},\ell_{1,r}}, \mathsf{tag})$, for $r = i+1, \ldots, q_1$.
        ii. If one of the double tags belongs to $\mathcal{X}$, then remove the pairs containing tag from S_i.
  3. For $d = 3, \ldots, \ell$: `# Filtering using clause` D_d
     (a) For $i = 1, \ldots, q_1$:
        i. For each $(\mathsf{tag}, \mathsf{ev}) \in S_i$:
           A. Compute $\mathsf{dtag}_j \leftarrow F(K^{\mathsf{x}}_{\ell_{1,i},\ell_{d,j}}, \mathsf{tag})$, for $j = 1, \ldots, q_d$.
           B. If one of $\mathsf{dtag}_1, \ldots \mathsf{dtag}_{q_d}$ belongs to $\mathcal{X}$ then, set $S \leftarrow S \setminus \{(\mathsf{tag}, \mathsf{ev})\}$.
  4. Return all second components appearing in $S = S_1 \cup S_2 \cup \ldots \cup S_{q_1}$. That is, parse S as $S = \{(\mathsf{tag}_1, \mathsf{ev}_1), \ldots, (\mathsf{tag}_{|S|}, \mathsf{ev}_{|S|})\}$ and return $\mathtt{ans} = \{\mathsf{ev}_1, \ldots, \mathsf{ev}_{|S|}\}$.
- $\mathtt{ans} \leftarrow \mathsf{CNFFilter.Resolve}(\mathtt{ans} = \{\mathsf{ev}_1, \ldots, \mathsf{ev}_{|\mathtt{ans}|}\}, \mathtt{msk} = (K^{\mathsf{p}}, K^{\mathsf{x}}, K^{\mathsf{enc}}, \mathtt{msk}^{\mathsf{p}}))$
  1. Return $\mathsf{Dec}(K^{\mathsf{enc}}, \mathsf{ev}_1), \ldots, \mathsf{Dec}(K^{\mathsf{enc}}, \mathsf{ev}_{|\mathtt{ans}|})$.

Fig. 2: Pseudocode for Construction $\mathsf{CNFFilter}$

size $|\mathsf{MM}[D_1]|$ that is most likely larger. We show that our scheme has better concrete server computation in our experiments in Section 5.

4.3 Formal Description of Leakage of CNFFilter

In this section, we give a formal description of the leakage for CNFFilter. We will utilize the partitioning of leakage into setup $\mathcal{L}_{\mathsf{Setup}}$, token $\mathcal{L}_{\mathsf{Token}}$ and search $\mathcal{L}_{\mathsf{Search}}$ leakage as done in ConjFilter.

Consider a multi-map $\mathsf{MM} = \{(\ell_t, \vec{v}_t)\}_{t\in[m]}$ and a CNF query sequence $Q = (\Phi^1, \ldots, \Phi^{|Q|})$, where $\Phi^p = (D_1^p \wedge \ldots \wedge D_{m^p}^p)$ consists of m^p clauses. The d-th clause D_d^p of Φ^p consists of q_d^p labels, $D_d^p = (\ell_{d,1}^p \vee \ldots \vee \ell_{d,q_d^p}^p)$.

Setup Leakage. As ConjFilter and CNFFilter have identical setup algorithms, the setup leakages are also identical. Thus, $\mathcal{L}_{\mathsf{Setup}}(\mathsf{MM}, Q) = N = \sum_{\ell,\ell'\in\mathcal{U}} |\mathsf{MM}[\ell] \cap \mathsf{MM}[\ell']|$.

Token Leakage. The token leakage consists of repetition patterns for both the decryption keys and PRF seeds for double tags. denoted by encryptionKeyRP and doubleTagSeedRP defined as follows. Each entry of encryptionKeyRP and doubleTagSeedRP will correspond to the decryption key or PRF seeds unique identifier. The token for the p-th query Φ^p contains one decryption key for each pair consisting of a label from the first clause and a label from the second clause. Therefore, for query Φ^p, $\mathsf{encryptionKeyRP}_p$ contains an entry $\mathsf{encryptionKeyRP}_p[i,j]$ for each $1 \le i \le q_1^p$ and $1 \le j \le q_2^p$. A repetition $\mathsf{encryptionKeyRP}_p[i,j] = \mathsf{encryptionKeyRP}_{p'}[i',j']$ occurs if and only if

$$\ell_{1,i}^p = \ell_{1,i'}^{p'} \text{ and } \ell_{2,j}^p = \ell_{2,j'}^{p'}.$$

In other words, the encryptionKeyRP tells us whether the first two clauses of two queries share two labels.

The token for the p-th query Φ^p contains one double-tag seed for each pair consisting of a label from the first clause and a label from a clause following the second clause. Therefore, for query Φ^p, $\mathsf{doubleTagSeedRP}_p$ contains an entry $\mathsf{doubleTagSeedRP}_p[i,d,j]$ for each $1 \le i \le q_1^p$, $1 \le d \le l^p$, and $1 \le j \le q_d^p$. A repetition $\mathsf{doubleTagSeedRP}_p[i,d,j] = \mathsf{doubleTagSeedRP}_{p'}[i',d',j']$ occurs if and only if

$$\ell_{1,i}^p = \ell_{1,i'}^{p'} \text{ and } \ell_{d,j}^p = \ell_{d,j'}^{p'}.$$

In other words, the doubleTagSeedRP tells us whether the first clauses and the d-th and d'-th clause of two queries share two labels.

Search Leakage. The execution of the Search algorithm reveals both tags and double tags. The tags are revealed after decrypting the response to the queries MM^{p}. The double tags are computed during filtering with the set $\mathcal{X}$. As both tags and double tags are pseudorandomly generated, they also leak repetition patterns.

In addition, double tags leak membership in $\mathcal{X}$ which is encoded in the matrices MX. Therefore, we have that

$$\mathcal{L}^{\mathsf{sr}}_{\mathsf{CNFFilter}}(\mathsf{MM}, Q) = (\mathsf{tagRP}, \mathsf{doubleTagRP}, \mathsf{MX}).$$

Let us now see what $\mathcal{L}^{\mathsf{sr}}_{\mathsf{CNFFilter}}$ tells us about MM. In computing the response to Φ^p, the number of tags obtained from each query to the underlying MM^{p} gives the volume of the 2-conjunction $\ell^p_{1,i} \wedge \ell^p_{2,j}$. In addition, observe that if the responses to 2-conjunction $(\ell^p_{1,i} \wedge \ell^p_{2,j})$ and to 2-conjunction $(\ell^{p'}_{1,i'} \wedge \ell^p_{2,j'})$ share a tag then it must be the case that $\ell^p_{1,i} = \ell^{p'}_{1,i'}$ and that there exists $v \in \mathsf{MM}[\ell^p_{1,i} \wedge \ell^p_{2,j} \wedge \ell^p_{2,j'}]$. Therefore, by counting the number of common tags between the responses to the two 2-conjunction one can obtain the volume of the 3-conjunction

$$\left(\ell^p_{1,i} \wedge \ell^p_{2,j}\right) \wedge \left(\ell^{p'}_{1,i'} \wedge \ell^p_{2,j'}\right) = \left(\ell^p_{1,i} \wedge \ell^p_{2,j} \wedge \ell^p_{2,j'}\right).$$

Clearly, tags appearing in the results of three or more 2-conjunctions give the volume of conjunctions with four or more labels.

In sums, we can say that the tag repetition pattern leaks the volume of 2-conjunctions, one for each query to MM^{p}, that can be combined to compute the volume of larger conjunctions.

The double-tag repetition pattern $\mathsf{doubleTagRP}$ can be computed from tagRP and $\mathsf{doubleTagSeedRP}$. Indeed, two double tags are equal iff they are obtained by applying the same double-tag seed to the same tag (except with negligible in λ probability). Therefore no further information is leaked by $\mathsf{doubleTagRP}$.

Finally, let us look at the double-tag membership in $\mathcal{X}$ pattern. For each query, the membership information MX^p for query Φ^p has a matrix $\mathsf{MX}^p_{i,j}$ for each pair of label $\ell^p_{1,i}$ of the first clause and label $\ell^p_{2,j}$ of the second clause. Matrix $\mathsf{MX}^p_{i,j}$ has a row for each tag tag that is obtained by decrypting the response to the query for $\mathsf{MM}^{\mathsf{p}}[\ell^p_{1,i} \wedge \ell^p_{2,j}]$ and a column for each double-tag seed dstag and $\mathsf{MX}^p_{i,j}[\mathsf{tag}, \mathsf{dstag}] = 1$ iff the corresponding double tag is found in $\mathcal{X}$. It is easy to see that the number of 1 in the column of double-tag seed for $\ell^p_{1,i}$ and $\ell^p_{1,r}$ gives the volume of 3-conjunction $(\ell^p_{1,i} \wedge \ell^p_{2,j} \wedge \ell^p_{1,r})$. Similarly the columns of the double-tag seed for $\ell^p_{1,i}$ and $\ell^p_{d,r}$ gives the volume of 3-conjunction $(\ell^p_{1,i} \wedge \ell^p_{2,j} \wedge \ell^p_{d,r})$. And, as before, the volume of 3-conjunctions can be combined together to obtain the volume of conjunction of size 4 or larger. In sums the membership in $\mathcal{X}$ gives the volume of conjunctions of size 3 or larger.

We denote the query leakage as the combination of token and search leakage $\mathcal{L}_{\mathsf{Query}} = (\mathcal{L}_{\mathsf{Token}}, \mathcal{L}_{\mathsf{Search}})$. We prove the following theorem in the full version.

Theorem 5. $\mathsf{CNFFilter}$ *is an adaptively* $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$*-secure encrypted Boolean multi-map scheme that supports CNF queries in the random oracle model.*

4.4 Comparing the leakage

We now compare the leakage of our construction with the one of BIEX (a description of BIEX may be found in the full version).

We start by considering the tags that are revealed by CNFFilter. Consider a CNF query with m clauses of the form $(\ell_{1,1} \vee \ldots \vee \ell_{1,q_1}) \wedge \ldots \wedge (\ell_{m,1} \vee \ldots \vee \ell_{m,q_m})$. CNFFilter reveals the tags for all values in $\mathsf{MM}[\ell_{1,i} \wedge \ell_{2,j}]$ for all label pairs of the form $(\ell_{1,i}, \ell_{2,j})$. On the other hand, BIEX reveals all plaintext tags appearing in $\mathsf{MM}[\ell_{1,i}]$ for all labels $\ell_{1,i}$. Additionally, it reveals all plaintext tags appearing in $\mathsf{MM}[\ell_{1,i} \wedge \ell_{j,k}]$ for all label pairs of the form $(\ell_{1,i}, \ell_{j,k})$ where $j \geq 2$.

Note that CNFFilter reveals double tags that do not exist in BIEX. The leakage reveals whether the double tag corresponding to label pair $(\ell_{1,i}, \ell_{k,l})$ and a value $v \in \mathsf{MM}[\ell_{1,i} \wedge \ell_{2,j}]$ appears in $\mathcal{X}$. Note that this is true if and only if $v \in \mathsf{MM}[\ell_{1,i} \wedge \ell_{2,j} \wedge \ell_{k,l}]$. Therefore, CNFFilter ends up leaking the volumes of 3-conjunctions of the form $\ell_{1,i} \wedge \ell_{2,j} \wedge \ell_{k,l}$ where $k \geq 3$. These are the only sets of PRF evaluations that are leaked by CNFFilter on top of the 2-conjunction results $\ell_{1,i} \wedge \ell_{2,j}$. Therefore, the base query set of leakage is $\mathsf{B}' \cup \{(b' \wedge \ell_{k,l}) \mid b' \in \mathsf{B}', 3 \leq k \leq m, l \in [q_k]\}$ where $\mathsf{B}' = \{(\ell_{1,i} \wedge \ell_{2,j}) \mid i \in [q_1], j \in [q_2]\}$.

On the other hand, the sets of PRF evaluations leaked by BIEX consist of queries from the set $\{\ell_{1,i} \mid i \in [q_1]\} \cup \{(\ell_{1,i} \wedge \ell_{j,k}) \mid i \in [q_1], 2 \leq j \leq m, k \in [q_j]\}$. This also turns out to be the base set of query leakage for BIEX.

It is easy to verify that the span of the base set of query leakage of CNFFilter is a subset of the span of the base set of query leakage of BIEX. First, BIEX leaks volumes for all singleton labels $\ell_{1,i}$ which CNFFilter doesn't leak, unless the query is a disjunction (recall that CNFFilter falls back to using BIEX in this case). Additionally, while CNFFilter only leaks 2-conjunctions of the form $\ell_{1,i} \wedge \ell_{2,j}$, BIEX leaks 2-conjunctions of the form $\ell_{1,i} \wedge \ell_{j,k}$ for all $j \geq 2$. Note that the 3-conjunctions $\ell_{1,i} \wedge \ell_{2,j} \wedge \ell_{k,l}$ leaked by CNFFilter are also leaked by BIEX since the server can compute this from the response sets of 2-conjunctions $\ell_{1,i} \wedge \ell_{2,j}$ and $\ell_{1,i} \wedge \ell_{k,l}$. Thus, it follows that CNFFilter does not leak more than BIEX.

As a caveat, we note that CNFFilter and BIEX leak volumes for the same set of queries in only two cases. The first case happens when the query is a disjunction of the form $\ell_1 \vee \ldots \vee \ell_q$, in which case CNFFilter falls back to the default implementation of BIEX. The other case happens when the query is a 2-conjunction of the form $\ell_1 \wedge \ell_2$. In every other case, CNFFilter leaks strictly less than BIEX.

5 Experiments

In this section, we present our experimental evaluation for our main construction, CNFFilter, that supports CNF queries with reduced volume leakage. We start by describing our experimental setup as well as the choice of parameters and primitives for our construction. Afterwards, we compare with the construction of BIEX described in [28].

Using the results of these experiments, we will try to answer the following question: *how do the concrete efficiency costs of our construction* CNFFilter *compare to the previous, state-of-the-art BIEX [28]?*

Note that we will use multipliers to describe efficiency improvements. If construction A is a 2x improvement over construction in B in computation, we mean that construction A uses half the computation compared to construction B.

5.1 Setup of Experiments

Our experiments are conducted using the identical machines for both the client and the server. The machines are Ubuntu PCs with 12 cores, 3.65 GHz Intel Xeon E5-1650 and 32 GB of RAM. All experimental results that are reported have standard deviations less than 10% of their average over 50 executions. All network costs are measured at the application layer. Both our client and server are implemented in C++ using the gRPC library [2].

Input Dataset. For our experiments, we utilize the Enron email dataset [34]. We parse the Enron email dataset using the Natural Language Toolkit (NLTK) in Python [3]. Before indexing the dataset, we perform canonicalization and stemming [40] using NLTK. Afterwards, we create a multi-map over the Enron email dataset mapping keywords to email identifiers. In our experiments, we will consider executing schemes over an input multi-map with a target number of values n. To obtain an input multi-map of size n from the Enron email dataset, we perform sampling in the following way. Pick emails uniformly at random and add them to the multi-map until there are at least n total keyword-identifier pairs in the multi-map.

Primitives. In all our experiments, we will utilize HMAC-SHA256 as our PRF with 16 byte keys. For our symmetric encryption scheme, we utilize AES in CTR mode with 16 byte keys. For the case of when encrypting pseudorandom values that will never repeat, we will utilize AES in CTR mode with a fixed IV. Our implementations utilize OpenSSL for both HMAC-SHA256 and AES. For the underlying standard encrypted multi-map of Section 2.3, we utilize the response-revealing 2Lev construction from [14] with parameters big block size $B = 100$ and small block size $b = 8$.

Selectivity of Clauses. In our experiments, we vary the selectivity of the first and second clauses in the CNF queries while fixing the selectivities of the remaining clauses. This is a reasonable setup as the search times of BIEX and CNFFilter depend mainly on the selectivity of the first and second clauses.

5.2 Implementation of BIEX [28]

The BIEX construction was presented in [28] along with an implementation in Java [1]. To provide a fair comparison with our C++ implementation of CNFFilter, we re-implement BIEX in C++ with the same underlying primitives as CNFFilter. All our reported results for BIEX will be using our C++ implementation. We note that the tags stored in the encrypted multi-map of BIEX will be the first 8 bytes of the HMAC-SHA256 output. As tags are pseudorandom and won't collide (except with small probability), we encrypt tags using AES-CTR mode with a fixed IV. All encryption and PRF keys used are 16 bytes long.

	100		500		1,000		5,000		10,000	
	CNFFilter	BIEX	CNFFilter	BIEX	CNFFilter	BIEX	CNFFilter	BIEX	CNFFilter	BIEX
100	$<$ **0.01**	$<$ 0.01	$<$ **0.01**	$<$ 0.01	$<$ **0.01**	$<$ 0.01	$<$ **0.01**	$<$ 0.01	$<$ **0.01**	$<$ 0.01
500	$<$ **0.01**	0.24	$<$ **0.01**	0.16	$<$ **0.01**	0.08	$<$ **0.01**	0.16	**0.04**	0.18
1000	$<$ **0.01**	1.28	$<$ **0.01**	1.22	$<$ **0.01**	1.24	**0.32**	1.30	**0.82**	1.36
5000	$<$ **0.01**	9.80	$<$ **0.01**	9.98	**0.64**	10.18	**3.01**	10.72	**5.01**	11.30
10000	$<$ **0.01**	21.84	**0.46**	20.34	**1.16**	21.98	**5.44**	22.36	**9.46**	22.76

Table 1: Microbenchmarks for the search time of CNFFilter and BIEX [28] on randomly chosen queries of the form $D_1 \wedge D_2 \wedge D_3$ where each D_i is a four label disjunction. The leftmost column and the topmost row denote the number of values associated with each label in the first and the second clause, respectively. The number of values associated with labels in D_3 are fixed to 10000. All search times are measured in milliseconds.

We did not implement their new underlying encrypted multi-map ZMF, filtering optimization or online cipher HBC1 [6], as they mainly improve the underlying encrypted multi-map which are used by both schemes in similar ways.

Compared to the Java implementation of BIEX [1], our C++ implementation of BIEX runs 20x faster than results reported in [28]. Recall that the server computation time depends on the selectivity of the first clause in the CNF query. For a query of the form $D_1 \wedge \ldots \wedge D_\ell$ over q distinct labels and each D_i is a disjunction, then BIEX search algorithm runs in time $O(q^2 \cdot |\mathsf{MM}[D_1]|)$. As the size of $\mathsf{MM}[D_1]$ grows, the server running time also grows as seen in Table 1.

5.3 Cost of CNFFilter

We also implement our construction CNFFilter in C++. The tags stored in the encrypted multi-map and the double tags stored in $\mathcal{X}$ will be the first 8 bytes of the HMAC-SHA256 output. As tags are pseudorandom and do not repeat (except with small probability), we encrypt tags using AES-CTR mode with a fixed IV. All encryption and PRF keys used in CNFFilter are 16 bytes long.

Recall that search computation time of CNFFilter depends on the selectivity of the conjunction of the first two clauses of a CNF query. For a CNF query of the form $D_1 \wedge \ldots \wedge D_\ell$ over q distinct labels, the running time of the search algorithm of CNFFilter grows in the size of $\mathsf{MM}[D_1 \wedge D_2]$. As $\mathsf{MM}[D_1 \wedge D_2]$ is a subset of $\mathsf{MM}[D_1]$, the running time of CNFFilter is expected to be faster than BIEX. Our experimental results in Table 1 confirm these expectations by showing that the search time of CNFFilter is at least 2x faster and may be more than 40x faster compared to BIEX for the same queries and the same input multi-map.

For communication, both CNFFilter and BIEX obtain optimal download communication complexity. In terms of upload communication costs (i.e. token size), CNFFilter requires smaller tokens compared to BIEX as shown in Figure 3. Recall that if the CNF query has q_1 labels in the first clause and q total labels overall, then BIEX executes $q_1 \cdot (q - q_1)$ queries to 2Lev. On the other hand, CNFFilter performs only $q_1 \cdot q_2$ 2Lev queries where q_2 is the number of labels in the second clause. The remaining operations in CNFFilter involve hashing and

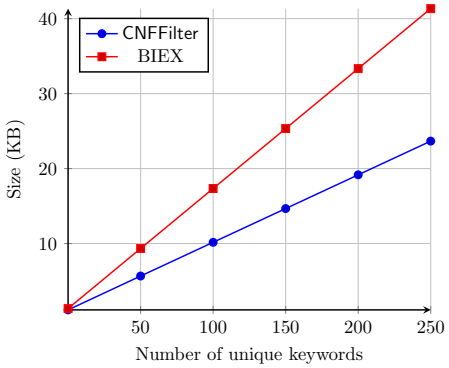

Fig. 3: Search token sizes of CNFFilter and BIEX [28] for 3-clause CNFs $D_1 \wedge D_2 \wedge D_3$ where the D_1 and D_2 contain 5 labels and the x-axis indicates the number of labels in D_3.

	Input Multi-Map Size in Number of Key-Value Pairs (n)									
	10,000		50,000		100,000		500,000		1,000,000	
	CNFFilter	BIEX	CNFFilter	BIEX	CNFFilter	BIEX	CNFFilter	BIEX	CNFFilter	BIEX
Storage Size (MB)	115	95	480	401	941	773	4,661	3,702	16,408	13,173
Setup Time (seconds)	3	2	13	8	24	13	137	72	662	308

Table 2: Storage and setup time of CNFFilter and BIEX [28].

checking membership in $\mathcal{X}$. These operations only require the client to send a PRF key of 16 byte size. This is around 60% of the token size of performing a query to 2Lev. Therefore, CNFFilter obtains smaller token sizes.

Finally, we consider the storage costs of CNFFilter and BIEX. Both schemes store an identical encrypted multi-map for all 2-conjunctions. In addition, CNFFilter must also store the set of hashes $\mathcal{X}$ that does not exist in BIEX. As a result, CNFFilter will have larger storage costs. However, the set $\mathcal{X}$ only consists of double tags of 8 byte length. As a result, $\mathcal{X}$ occupies much smaller space compared to the storage of the encrypted multi-map. This is observed in Table 2 that shows CNFFilter only incurs a 20-25% increase in storage over BIEX, which seems reasonable given the leakage, communication and server computation improvements.

6 Conclusions

In this work, we continue work on designing encrypted Boolean multi-maps. Our new construction CNFFilter mitigates volume leakage better than all previous works while simultaneously achieving optimal communication and worst-case sublinear search times. In terms of volume leakage reduction, CNFFilter substantially improves upon the previous constructions.

References

1. Clusion. https://github.com/orochi89/Clusion.
2. gRPC - an RPC library and framework. https://github.com/grpc/grpc.
3. Natural language toolkit. http://nltk.org.
4. SPAR Pilot Evaluation. MIT Lincoln Laboratory, 2015.
5. G. Asharov, G. Segev, and I. Shahaf. Tight tradeoffs in searchable symmetric encryption. In *CRYPTO '18*, 2018.
6. M. Bellare, A. Boldyreva, L. Knudsen, and C. Namprempre. On-line ciphers and the hash-CBC constructions. *Journal of Cryptology*, 2012.
7. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO '07*, 2007.
8. L. Blackstone, S. Kamara, and T. Moataz. Revisiting leakage abuse attacks. In *NDSS '20*, 2020.
9. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT '04*, 2004.
10. R. Bost. Sophos: Forward secure searchable encryption. In *CCS '16*, 2016.
11. R. Bost and P.-A. Fouque. Security-efficiency tradeoffs in searchable encryption – lower bounds and optimal constructions. In *PoPETS '19*, 2019.
12. R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *CCS '17*, 2017.
13. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS '15*, 2015.
14. D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS '14*, 2014.
15. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13*, 2013.
16. D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *EUROCRYPT '14*, 2014.
17. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT '10*, pages 577–594. Springer, 2010.
18. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 2011.
19. I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *CRYPTO '18*, 2018.
20. I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *SIGMOD '17*, 2017.
21. I. Demertzis, R. Talapatra, and C. Papamanthou. Efficient searchable encryption through compression. *PVLDB '18*, 2018.
22. S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *ESORICS '15*, 2015.
23. C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC '09*.
24. O. Goldreich and R. Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3), 1996.
25. P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE S&P '19*.

26. P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *CCS '18*.
27. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS '12*, 2012.
28. S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT '17*, 2017.
29. S. Kamara and T. Moataz. SQL on structurally-encrypted databases. In *ASIACRYPT '18*, 2018.
30. S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *EUROCRYPT 2019*, 2019.
31. S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In *CRYPTO '18*, 2018.
32. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS '12*, 2012.
33. G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *CCS '16*, 2016.
34. B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *ECML PKDD*, 2004.
35. M. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *IEEE S&P '18*, 2018.
36. I. Miers and P. Mohassel. Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. In *NDSS '17*, 2017.
37. V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind Seer: A scalable private DBMS. In *IEEE S&P '14*, 2014.
38. S. Patel, G. Persiano, and K. Yeo. Leakage cell probe model: Lower bounds for key-equality mitigation in encrypted multi-maps. In Crypto '20.
39. S. Patel, G. Persiano, K. Yeo, and M. Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *CCS '19*.
40. M. Porter. The Porter stemming algorithm. https://tartarus.org/martin/PorterStemmer/.
41. D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P '00*, 2000.
42. E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS '14*, 2014.
43. 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*, pages 707–720, 2016.

A Overview of BIEX [28]

In this section, we present an overview of the algorithm utilized by BIEX to handle CNF queries. We note that the setup algorithm of BIEX is similar to both ConjFilter and CNFFilter. BIEX's setup algorithm will create and outsource the storage of $\mathsf{EMM}^{\mathsf{p}}$ to the server.

Consider a CNF query of the form $D_1 \wedge \ldots \wedge D_\ell$ where each $D_i = (\ell_{i,1} \vee \ldots \vee \ell_{i,q_i})$ is a disjunction over q_i labels. We denote the total number of labels in the CNF query as q. As the first step, BIEX computes the result set for the first clause, $\mathsf{MM}[D_1]$. Since D_1 is a disjunction, BIEX leaks the volumes of singleton queries for all labels appearing in D_1. The response set $\mathsf{MM}[D_1]$ is computed such that it is partitioned into parts $P_1, \ldots, P_{q_1}$ where the values in each part P_i are hashed by a PRF key depending only on label $\ell_{1,i}$. Next, the algorithm filters each part P_i in the following way. To compute $P_i \wedge D_2 \wedge \ldots \wedge D_\ell$, the algorithm computes the responses of $(q_2+q_3+\ldots+q_\ell)$ 2-conjunctions of the form $\mathsf{MM}[\ell_{1,i} \wedge \ell]$ where ℓ is a label that appears in any of $D_2, \ldots, D_\ell$. The responses for all these 2-conjunctions will also be hashed under a PRF key depending only on label $\ell_{1,i}$. Afterwards, the server computes the intersection of P_i and all $(q_2 + \ldots + q_\ell)$ responses of 2-conjunctions to compute the set $P_1 \wedge D_2 \wedge \ldots D_\ell$. After filtering all q_i parts and taking the union, the server successfully computes the answer.

A.1 Volume Leakage of BIEX

We now describe the volume leakage of BIEX. As stated earlier, it is not known how to handle disjunctive queries without revealing the volumes of singleton label queries. As a result, the volumes of q_1 singleton queries for all labels appearing in the first clause D_1 are leaked. Furthermore, the filtering of each of the q_1 parts requires computing the response and, thus, revealing the volume of $(q_2 + \ldots + q_\ell)$ 2-conjunctive queries. Therefore, the volume of $q_1 \cdot (q_2 + \ldots + q_\ell)$ 2-conjunctive queries must also be revealed by their algorithm. Note, there are many sets of hashed values under the same key. For example, P_i and any of the sets $\mathsf{MM}[\ell_{1,i} \wedge \ell]$ are hashed under the same keys. Therefore, the server may compute arbitrary intersections. Using these ideas, the server may compute the volume of any 3-conjunctions of the form $\mathtt{a} \wedge \mathtt{b} \wedge \mathtt{c}$ where $\mathtt{a}$ must appear in the first clause and both $\mathtt{b}$ and $\mathtt{c}$ may appear in any clause except the first clause. Therefore, the server learns the volumes of $q_1 \cdot \binom{q-q_1}{2}$. More generally, it can be shown that $q_1 \cdot \binom{q-q_1}{k-1}$ volumes for k-conjunctions are leaked by BIEX for any $k \geq 2$.

B Proof of Security for ConjFilter

We present a proof that SIM described in Figure 4 is a simulator for ConjFilter by considering the following sequence of games.

1. Game_0 is the real game with input MM and query sequence Q.

Non-Adaptive Simulator SIM takes as input leakage $\mathcal{L}_{\mathsf{ConjFilter}}(\mathsf{MM}, Q) = (N, (q, q_1, \ldots, q_q, \mathsf{encryptionKeyRP}, \mathsf{doubleTagSeedRP}), (L_1, \ldots, L_q, \mathsf{tagRP}, \mathsf{doubleTagRP}, \mathsf{MX}_1, \ldots, \mathsf{MX}_q))$.

1. Reconstruct qeq from $\mathsf{encryptionKeyRP}$ and randomly select $K^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$.
2. Let $v^\star \in \mathcal{V}$ be an arbitrarily chosen value from the value universe.
3. Construct an array $\mathsf{Tag}[1, \ldots, T]$ of T randomly selected tags from $\{0,1\}^\lambda$, where T is the number of different tags as read from tagRP.
4. Construct an array $\mathsf{DK}[1, \ldots, D]$, where D is the number of different decryption keys as read from $\mathsf{encryptionKeyRP}$, and set each entry equal to $\perp$.
5. Initialize $\mathsf{vals} \leftarrow \emptyset$.
6. Split tagRP into q sequences $\mathsf{tagRP}_1, \ldots, \mathsf{tagRP}_q$ of lengths $L_1, \ldots, L_q$.
7. For $i = 1, \ldots, q$:
   (a) Set $r \leftarrow \mathsf{encryptionKeyRP}[i]$.
   (b) If $\mathsf{DK}[r] \neq \perp$ then `// repeated query`
      i. Set $\mathsf{resp}_i \leftarrow \mathsf{resp}_r$ and continue to next i.
   (c) Initialize $\mathsf{resp}_i = ()$. `// new query`
   (d) Randomly select $\mathsf{DK}[r] \leftarrow \{0,1\}^\lambda$.
   (e) For $l = 1, \ldots, L_i$
      i. Set $\mathsf{tag}_{i,l} \leftarrow \mathsf{Tag}[\mathsf{tagRP}_i[l]]$ and compute $\mathsf{etag}_{i,l} = \mathsf{Enc}(DK[i], \mathsf{tag}_{i,l})$.
      ii. Compute $\mathsf{ev}_{i,l} = \mathsf{Enc}(K^{\mathsf{Enc}}, v^\star)$.
      iii. Append $(\mathsf{etag}_{i,l}, \mathsf{ev}_{i,l})$ to resp_i and to vals
8. For $i = 1, \ldots, N - T$
   (a) Randomly select $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and set $\mathsf{etag} = \mathsf{Enc}(K^{\mathsf{Enc}}, \mathsf{tag})$.
   (b) Compute $\mathsf{ev} = \mathsf{Enc}(K^{\mathsf{Enc}}, v^\star)$ and append $(\mathsf{etag}, \mathsf{ev})$ to vals.
9. Run Simulator $\mathsf{SIM}^{\mathsf{p}}$ on input $(N, \mathsf{vals}, \mathsf{qeq}, \mathsf{resp})$ to obtain $\mathsf{EMM}^{\mathsf{p}}$ and $\mathtt{tok}_1^{\mathsf{p}}, \ldots, \mathtt{tok}_q^{\mathsf{p}}$.
10. Construct an array $\mathsf{Seed}[1, \ldots, S]$ of randomly selected double-tag seeds from $\{0,1\}^\lambda$, where S is the number of different double-tag seeds as read from $\mathsf{doubleTagSeedRP}$.
11. Split $\mathsf{doubleTagSeedRP}$ into $\mathsf{doubleTagSeedRP}_1, \ldots, \mathsf{doubleTagSeedRP}_q$ of length $q_1 - 2, \ldots, q_q - 2$.
12. Set $\mathcal{X} = \emptyset$. `// Constructing` $\mathcal{X}$
13. For $i = 1, \ldots, q$:
   (a) For $l = 1, \ldots, L_i$:
      i. Set $r \leftarrow \mathsf{tagRP}_i[l]$ and $\mathsf{tag} \leftarrow \mathsf{Tag}[r]$.
      ii. For $d = 3, \ldots, q$
         - $s \leftarrow \mathsf{doubleTagSeedRP}_i[d]$ and $K^{\mathsf{x}} \leftarrow \mathsf{Seed}[s]$.
         - if $\mathsf{MX}_i[l][d] = 1$, set $\mathcal{X} = \mathcal{X} \cup \{F(K^{\mathsf{x}}, \mathsf{tag})\}$.
14. For $i = 1, \ldots, q$: `// Constructing the tokens`
   (a) Set $\mathtt{tok}^i = (\mathtt{tok}_i^{\mathsf{p}})$, $r \leftarrow \mathsf{encryptionKeyRP}[i]$ and append $\mathsf{DK}[r]$ to $\mathtt{tok}^i$.
   (b) For $d = 3, \ldots, q$
      i. Set $s \leftarrow \mathsf{doubleTagSeedRP}_i[d-2]$ and append $\mathsf{Seed}[s]$ to $\mathtt{tok}^i$.
15. Pad $\mathcal{X}$ with a sufficient number of random elements of $\{0,1\}^\lambda$ so to reach cardinality N and then randomly permute $\mathcal{X}$.
16. Set $\mathsf{eBMM} = (\mathsf{EMM}^{\mathsf{p}}, \mathcal{X})$ and output $(\mathsf{eBMM}, \mathtt{tok}^1, \ldots, \mathtt{tok}^q)$.

Fig. 4: Pseudocode for Simulator SIM

2. Game_1 computes EMM^p and the tokens $\mathtt{tok}^{\mathsf{P}}$ using the simulator $\mathsf{SIM}^{\mathsf{P}}$ on input the leakage computed using MM and Q. The output of Game_1 is indistinguishable from the output of Game_0 by the properties of $\mathsf{SIM}^{\mathsf{P}}$.
3. Game_2 differs from Game_1 in the computation of ev_v at Step 4(d)ii of ConjFilter.Setup: ev_v is computed as the encryption of a fixed and arbitrarily chosen $v^\star \in \mathcal{V}$. The output of Game_2 is indistinguishable from the output of Game_1 by the IND-CPA security of the encryption scheme.
4. Game_3 differs from Game_2 as, at Step 4a of ConjFilter.Setup, the tag seeds are randomly chosen and then, at Step 4b, encryption keys are randomly chosen. The output of Game_3 is indistinguishable from the output of Game_2 by the pseudo-randomness of F.
5. Game_4 differs from Game_3 in the computation of $\mathsf{tag}_{\mathsf{a},v}$ at Step 4(d)i of ConjFilter.Setup as it is chosen at random and not computed pseudo-randomly. The output of Game_4 is indistinguishable from the output of Game_3 by the pseudo-randomness of F.
6. In Game_5, the encrypted tags that are not decrypted by any of the $|Q|$ invocations of Search are computed by encrypting a fixed and arbitrarily chosen tag $\mathsf{tag}^\star$ using a fixed and arbitrarily chosen encryption key $K^\star$. The output of Game_5 is indistinguishable from the output of Game_4 by the IND-CPA security and the key-oblivious property of the encryption scheme.
7. In Game_6, the encrypted tags that are decrypted by one of the $|Q|$ invocations of Search are computed by selecting the tags according to the tag repetition pattern tagRP. The output of Game_6 is identical to the output of Game_5.
8. In Game_7, the double-tag seeds are chosen at random. The output of Game_7 is indistinguishable from the output of Game_6 by the pseudo-randomness of F.
9. In Game_8, the double-tag seeds at Step 3a of ConjFilter.Token are chosen according to the double-seed repetition pattern. The output of Game_8 is identical to the output of Game_7.
10. In Game_9, $\mathcal{X}$ is computed by first applying double-tag seeds from the tokens to the tags in the responses of the queries and then by adding sufficiently many random values from $\{0,1\}^\lambda$ until $\mathcal{X}$ has cardinality N. The output of Game_9 is indistinguishable from the output of Game_8 by the pseudo-randomness of F.
    Finally, observe that the output of Game_9 is identical to the output of SIM.

The above simulator works for a non-adaptive adversary. We present the modifications necessary to obtain security against an adaptive adversary assuming the underlying encrypted multi-map sEMM is itself adaptively secure and that F is modeled as a random oracle. Note that the assumption of random oracles as well as their programmability are required for adaptive security by previous works [15,28] as well.

In the adaptively secure version of ConjFilter, we replace the IND-CPA encryption algorithm with the following *equivocable* encryption algorithm. $\mathsf{Enc}(K, m)$ consists of randomly selecting $r \leftarrow \{0,1\}^\lambda$, computing $h = F(K \mid\mid r)$ and outputting the ciphertext consisting of $(r, h \oplus m)$. Decryption of (c_1, c_2) using key

K is straightforward. It is easy to see that, if the random oracle F can be programmed, then every ciphertext (c_1, c_2) can be *equivocated* with respect to any decryption key K'; that is, for any K', (c_1, c_2) can be opened to be the encryption with respect to K' of any plaintext m'. This property will be crucial to prove adaptive security of the construction.

We next give a high-level view of the adaptive simulator aSIM for $\mathsf{ConjFilter}$ that uses as a subroutine the adaptive simulator $\mathsf{aSIM}^{\mathsf{p}}$ for sEMM. We remind the reader that simulator $\mathsf{aSIM}^{\mathsf{p}}$ first receives the setup leakage, that is the set vals of values on the underlying multi-maps MM^{p}, and then outputs the simulated $\mathsf{EMM}^{\mathsf{p}}$. Subsequently, for the i-th query to $\mathsf{EMM}^{\mathsf{p}}$, $\mathsf{aSIM}^{\mathsf{p}}$ receives the leakage $(\mathsf{qeq}_i, \mathsf{resp}_i)$ for the i-th query and outputs the token $\mathtt{tok}_i^{\mathsf{p}}$.

aSIM starts by receiving the setup leakage $\mathcal{L}^{\mathsf{st}}_{\mathsf{ConjFilter}}(\mathsf{MM})$ consisting of N, the size of $\mathcal{X}$ and of MM^{p}. It constructs vals by randomly selecting N ciphertexts of the equivocable encryption scheme described above and feeds it to $\mathsf{aSIM}^{\mathsf{p}}$ that returns MM^{p}. The set $\mathcal{X}$ instead is constructed by randomly selecting N random elements from $\{0,1\}^\lambda$ and eBMM is set equal to $(\mathsf{MM}^{\mathsf{p}}, \mathcal{X})$. Subsequently, aSIM sets up some local data structures to be used in the simulation: an array Tag for storing tags, an array DTag for storing information about the double tag, an array DK for storing decryption keys, and an array Seed for storing double-tag seeds. The arrays are initially empty.

For each query Φ^i, aSIM receives the leakage $\mathcal{L}^{\mathsf{q}}(\mathsf{MM}, \Phi^1, \ldots, \Phi^i)$ consisting of the length q_i, the decryption key repetition pattern $\mathsf{encryptionKeyRP}_i$, the double-tag seed repetition pattern $\mathsf{doubleTagSeedRP}_i$, length L_i, the tag repetition pattern tagRP_i, and the matrix MX_i of the membership to $\mathcal{X}$. First of all, aSIM constructs the leakage $(\mathsf{qeq}_i, \mathsf{resp}_i)$ needed by $\mathsf{aSIM}^{\mathsf{p}}$ to produce the token $\mathtt{tok}_i^{\mathsf{p}}$. Let $r = \mathsf{encryptionKeyRP}_i$. If $\mathsf{DK}[r] \neq \emptyset$, then it means that the i-th query Φ^i gives rise to the same query as Φ^r for MM^{p} and its response leakage resp_i coincides with resp_r. Moreover, the query equality pattern qeq_i can be computed by extending qeq_r to include the queries from the r-th to the $(i-1)$-st. Consider now the case $\mathsf{DK}[r] = \emptyset$; this implies that the query to MM^{p} generated by Φ^i differs from all previous ones. In this case aSIM randomly selects a new decryption key from $\{0,1\}^\lambda$ and stores it as $\mathsf{DK}[r]$. Then by looking at the tag repetition pattern tagRP_i of length L_i it identifies the tags that are going to appear in resp_i. Some of the tags have already appeared in previous responses and will be found in the array Tag; some instead appear for the first time and are thus chosen at random and stored in Tag. Once the set $\mathsf{tag}_{i,l}$, for $i = 1, \ldots, L_i$, has been identified, aSIM selects L_i still unused ciphertexts from vals and programs the random oracle so that the ciphertexts can be equivocated with key $\mathsf{DK}[r]$ to open as $\mathsf{tag}_{i,1}, \ldots, \mathsf{tag}_{i,L_i}$. The L_i ciphertexts constitute leakage resp_i and the qeq_i encodes the fact that the query differs from all the previous one. Once the leakage $(\mathsf{qeq}_i, \mathsf{resp}_i)$ is constructed, the token $\mathtt{tok}_i^{\mathsf{p}}$ is obtained by feeding it to $\mathsf{aSIM}^{\mathsf{p}}$.

Let us now see how the rest of the token $\mathtt{tok}^i$ is constructed. Clearly, key $\mathsf{DK}[r]$ is added to the token. For the double-tag seeds that are part of the token, aSIM uses a strategy similar to the one used for the tags. Specifically, aSIM reads

from the double-tag seed repetition pattern $\mathsf{doubleTagSeedRP}_i$ the seeds to be used in the token for Φ^i. As before, if the corresponding entry in Seed is empty, a new seed is selected and stored; otherwise, aSIM uses the seed found in the array. Then, for each double tag that according to MX_i is found in $\mathcal{X}$, aSIM checks from DTag if the double tag has already appeared. If a new double tag is needed then aSIM selects it at random, stores in DTag, and programs the random oracle so that the evaluation of double tag using the tag and double-tag seed gives the selected double tag.

C Simulator for CNFFilter

We start by describing the simulator SIM for $\mathsf{CNFFilter}$ that uses the simulator $\mathsf{SIM}^{\mathsf{p}}$ for the encrypted multi-map sEMM used as building block by $\mathsf{CNFFilter}$.

Simulator SIM receives in input the leakage $\mathcal{L}_{\mathsf{CNFFilter}}(\mathsf{MM}, Q)$ for a multi-map MM and a sequences of queries and constructs the leakage $(N, \vec{v}, \mathsf{qeq}, \mathsf{resp})$ for $\mathsf{SIM}^{\mathsf{p}}$. This is achieved similarly to what is done by the simulator for $\mathsf{ConjFilter}$. Once the leakage for $\mathsf{SIM}^{\mathsf{p}}$ is obtained, SIM feeds it to $\mathsf{SIM}^{\mathsf{p}}$ and obtains the tokens $\mathtt{tok}^{\mathsf{p}}_{p,i,j}$ for each query Φ^p of Q, and every $1 \leq i \leq q_1^p$ and and every $1 \leq j \leq q_2^p$, where q_1^p and q_2^p are, respectively, the number of labels in the first clause and in the second clause of Φ^p.

The set $\mathcal{X}$ is constructed by randomly generating enough double-tag seeds (the right number is read from the appropriate repetition pattern $\mathsf{doubleTagSeedRP}$) and by applying them to the tags that are obtained from the decryption of the encrypted tag in the executions of the $\mathsf{CNFFilter.Search}$ algorithm (which tags are releaved by $\mathsf{CNFFilter.Search}$ is specified by the tag repetition pattern tagRP). Next, SIM adds to $\mathcal{X}$ only the double tags that are specified to belong to $\mathcal{X}$ by MX. Finally, the set $\mathcal{X}$ is padded to size N by adding enough randomly chosen double tags.

Let us now describe how SIM constructs the token query for Φ^p. We have already described the generation of the tokens for the $q_1^p \cdot q_2^p$ queries to MM^{p}. To these, SIM adds the double-tag seeds generated at the previous step for the construction of $\mathcal{X}$ as specified by the repetition pattern $\mathsf{doubleTagSeedRP}$.

If we instead model F as a random oracle, it is possible to utilize an adaptively secure sEMM scheme and to modify $\mathsf{CNFFilter}$ so to use an equivocable encryption scheme based on F, similarly to what done for $\mathsf{ConjFilter}$, thus obtaining an adaptively secure construction with leakage $\mathcal{L}_{\mathsf{CNFFilter}}$.