TWo-IN-one-SSE: Fast, Scalable and Storage-Efficient Searchable Symmetric Encryption for Conjunctive and Disjunctive Boolean Queries

We present the proofs of the lemmas and theorems presented in the main paper here.

1 Adaptive Security of SSE

The adaptive security of any SSE scheme is parameterized by a leakage function

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

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

Algorithm 1 Experiment $\mathbf{Real}^{\mathsf{SSE}}(\lambda, Q)$

function $\mathbf{Real}^{\mathsf{SSE}}(\lambda, Q)$
  $N \leftarrow \mathbf{Adv}(\lambda)$
  $(\mathsf{sk}, \mathsf{st}_0, \mathbf{EDB}_0) \leftarrow \textsc{Setup}(\lambda, N)$
  for $k \leftarrow 1 \text{to} Q$ **do**
    Let $q_k \leftarrow \mathbf{Adv}(\lambda, \mathbf{EDB}_{k-1}, \tau_1, \ldots, \tau_{k-1})$
    Let $(\mathsf{st}_k, \mathbf{EDB}_k, \mathbf{DB}(q_k)) \leftarrow$
      $\textsc{Search}(\mathsf{sk}, \mathsf{st}_{k-1}, q_k; \mathbf{EDB}_{k-1})$
    Let τ_k denote the view of the adversary after
      the k^{th} query
  $b \leftarrow \mathbf{Adv}(\lambda, \mathbf{EDB}_Q, \tau_1, \ldots, \tau_Q)$
  return b

Informally, an SSE scheme is adaptively secure with respect to a leakage function $\mathcal{L}$ if the adversarial server provably learns no more information about **DB** other than that encapsulated by $\mathcal{L}$. Formally, an SSE scheme is said to be adaptively secure with respect to a leakage function $\mathcal{L}$ if for any stateful PPT adversary

Algorithm 2 Experiment $\mathbf{Ideal}^{\mathsf{SSE}}(\lambda, Q, \mathcal{L})$

function $\mathbf{Ideal}^{\mathsf{SSE}}(\lambda, Q, \mathcal{L})$
  Parse the leakage function $\mathcal{L}$ as:
  $\mathcal{L} = \left(\mathcal{L}^{\textsc{Setup}}, \mathcal{L}^{\textsc{Search}}\right)$.
  $(\mathsf{st}_{\textsc{Sim}}, \mathbf{EDB}_0) \leftarrow \textsc{SimSetup}(\mathcal{L}^{\textsc{Setup}}(\lambda, N))$
  for $k \leftarrow 1$ to Q **do**
    Let $q_k \leftarrow \mathbf{Adv}(\lambda, \mathbf{EDB}_{k-1}, \tau_1, \ldots, \tau_{k-1})$
    Let $(\mathsf{st}_{\textsc{Sim}}, \mathbf{EDB}_k, \tau_k) \leftarrow \textsc{SimSearch}$
      $(\mathsf{st}_{\textsc{Sim}}, \mathcal{L}^{\textsc{Search}}(q_k); \mathbf{EDB}_{k-1})$
    Let τ_k denote the view of the adversary after
      the k^{th} query
  $b \leftarrow \mathbf{Adv}(\lambda, \mathbf{EDB}_Q, \tau_1, \ldots, \tau_Q)$
  return b

$\mathbf{Adv}$ that issues a maximum of $Q = \mathsf{poly}(\lambda)$ queries, there exists a stateful probabilistic polynomial-time simulator $\textsc{Sim} = (\textsc{SimSetup}, \textsc{SimSearch})$ such that the following holds:

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

where the "real" experiment $\mathbf{Real}^{\mathsf{SSE}}$ and the "ideal" experiment $\mathbf{Ideal}^{\mathsf{SSE}}$ are as described in Algorithm 1 and Algorithm 2 (in Appendix).

2 Proof of Lemma 3.1 (Already presented in main body)

Proof. In the formal proof of Lemma 3.1 we show that, each mkw constructed following the description of Lemma 3.1 covers each w in q (the $\mathbf{DB}(q)$ part) and other ws that are not in q are filtered out (the adjacent intersection in Lemma 3.1).

We start with the following conjunctive expression of mkws of q_{mkw} for a particular query q as given in the Lemma 3.1.

$$\begin{aligned}\mathbf{DB}(q_{\mathsf{mkw}}) &= \mathbf{DB}(\mathsf{mkw}_{i_1,j_1} \wedge \ldots \wedge \mathsf{mkw}_{l_n,j_n}) \\ &= \bigcap_{k=0}^{n} \mathbf{DB}(\mathsf{mkw}_{i_k,j_k})\end{aligned} \tag{1}$$

By the definition of meta-keywords (Definition 3.1), the following relation holds.

$$\mathbf{DB}(\mathsf{mkw}_{l_k,j_k}) = \bigcup_{\ell \in [N] \setminus [l_k,j_k]} \mathbf{DB}(\mathsf{w}_\ell)$$

We rewrite Equation (1) in the following way.

$$
\begin{aligned}
\mathbf{DB}(q_{\mathsf{mkw}}) &= \bigcap_{k=0}^{n} \mathbf{DB}(\mathsf{mkw}_{i_k,j_k}) \qquad (2)\\
&= \bigcap_{k=0}^{n} \Big(\bigcup_{\ell \in [N]\setminus[l_k,j_k]} \mathbf{DB}(\mathsf{w}_\ell) \Big)\\
&= \bigcap_{k=0}^{n} \Big(\bigcup_{r\in[1,n]} \mathbf{DB}(\mathsf{w}_{l_r}) \cup \bigcup_{\substack{\ell \in [n]\setminus(\{[l_k+1,j_k-1]\}\\ \cup\{l_r : r\in[0,n]\})}} \mathbf{DB}(\mathsf{w}_\ell) \Big)\\
&= \bigcup_{k\in[0,n]} \mathbf{DB}(\mathsf{w}_{l_k}) \cup \bigcap_{k=0}^{n} \Big(\bigcup_{\substack{\ell \in [N]\setminus(\{[l_k+1,j_k-1]\}\\ \cup\{l_r : r\in[0,n]\})}} \mathbf{DB}(\mathsf{w}_\ell) \Big)\\
&= \mathbf{DB}(q) \cup \Big(\bigcap_{k=0}^{n} \Big(\bigcup_{\substack{\ell \in [N]\setminus(\{[l_k+1,j_k-1]\}\\ \cup\{l_r : r\in[0,n]\})}} \mathbf{DB}(\mathsf{w}_\ell) \Big) \Big)
\end{aligned}
$$

Observe that, the union inside the right hand term in Equation (2) keeps all ws except a stretch of ws (from index l_k+1 to $l_{k+1}-1$) for each value of k, inside the outer intersection of $n+1$ terms. Since the intersection of these unions reduces to a small but finite set of ids, the following relation holds.

$$\mathbf{DB}(q) \subseteq \mathbf{DB}(q_{\mathsf{mkw}})$$

We validated this through experimental evaluations (presented in main paper). The intersection on the right of $\mathbf{DB}(q)$ in the last step of Equation (2) (in Appendix) essentially introduces the spurious ids in the result set. □

3 Proof of Theorem 4.1 (Correctness of TWINSSE$_{\text{Basic}}$ and TWINSSE)

The proof of correctness for TWINSSE$_{\text{Basic}}$ (and TWINSSE as well) follows from the correctness of CSSE. The correctness of CSSE ensures that a conjunctive query $q = \mathsf{w}_1 \wedge \cdots \wedge \mathsf{w}_n$ over an encrypted database satisfies the following relations.

$$\mathbf{EDB} = \text{CSSE.}\textsc{Setup}(\mathbf{DB}))$$

$$\mathbf{DB}(\mathsf{w}_1) \cap \cdots \cap \mathbf{DB}(\mathsf{w}_n) = \text{CSSE.}\textsc{Search}(q, \mathbf{EDB}))$$

We state the proof for TWINSSE$_{\text{Basic}}$ first. Then we show that this can be simply extended to main TWINSSE scheme (the final bucketized version).

Proof for TWINSSE$_{\text{Basic}}$. Proof of the TWINSSE$_{\text{Basic}}$ directly follows from the proof of Lemma 3.1. Consider a disjunctive query q as stated below.

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

The equivalent conjunctive expression of meta-keywords can be expressed as below.

$$q_{\mathsf{mkw}} = \mathsf{mkw}_{i_0,j_0} \wedge \mathsf{mkw}_{i_1,j_1} \wedge \ldots \wedge \mathsf{mkw}_{i_n,j_n}$$

We write the following relation from Lemma 3.1.

$$\mathbf{DB}(q_{\mathsf{mkw}}) = \mathbf{DB}(q) \cup \Big(\bigcap_{k=0}^{n} \Big(\bigcup_{\substack{\ell \in [N] \setminus (\{[l_k+1, j_k-1]\} \\ \cup \{l_r : r \in [0,n]\})}} \mathbf{DB}(\mathsf{w}_\ell) \Big) \Big)$$

It easy to notice from the above equation that $\mathbf{DB}(q) \subseteq \mathbf{DB}(q_{\mathsf{mkw}})$. Hence, all ids of the actual result set of disjunctive query q is included in the result set obtained from the query using TWINSSE$_{\text{Basic}}$.Search, which proves the correctness of the TWINSSE$_{\text{Basic}}$.

Proof of TWINSSE. Recall from Section 4 (in main paper), that in TWINSSE all ws from Δ are partitioned into n_{B} buckets of uniform size, and we execute the basic meta-keyword generation method developed in TWINSSE$_{\text{Basic}}$ over each partition independently. Only those partitions with query meta-keywords are accessed during search.

Assume that the dictionary of ws - Δ is partitioned in the following way,

$$\Delta = \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_{n_{\text{B}}}$$

where n_{B} is the number of buckets and each bucket Δ_k can be expressed in the following way.

$$\Delta_k = \{\mathsf{w}_{(k-1)n'+1}, \mathsf{w}_{(k-1)n'+2}, \cdots, \mathsf{w}_{kn'}\}$$

The number of ws in each bucket is denoted by n'. The set of mkws in each bucket $\mathcal{W}_k$ are represented by the $\mathcal{S}_{\mathsf{mkw},k}$. The TWINSSE$_{\text{Basic}}$ is executed over each of these bucket individually to generate the encrypted database.

The query expression follows from the TWINSSE construction with the above structure (discussed in Section 4 of main paper).

$$\mathbf{DB}(q_{\mathsf{mkw}}) = \bigcup_{k \in [n_{\text{B}}]} \mathbf{DB}(q_{\mathsf{mkw},k})$$

and the actual query q can be partioned in the following way.

$$q = \bigvee_{k \in [n_{\text{B}}]} q_k$$

We expand the above expression to individual buckets.

$$
\begin{aligned}
\mathbf{DB}(q_{\mathsf{mkw}}) &= \bigcup_{k\in[n_{\mathrm{B}}]} \mathbf{DB}(q_{\mathsf{mkw},k}) \\
&= \bigcup_{k\in[n_{\mathrm{B}}]} \Big(\mathbf{DB}(q_k) \cup \Big(\bigcap_{k=0}^{|q_k|}\Big(\bigcup_{\substack{\ell\in[|\mathcal{W}_k|]\setminus \\ (\{[l_k+1,j_k-1]\} \\ \cup\{l_r : r\in[0,|q_k|]\})}} \mathbf{DB}(\mathsf{w}_\ell)\Big)\Big)\Big) \\
&= \bigcup_{k\in[n_{\mathrm{B}}]} \mathbf{DB}(q_k) \cup \bigcup_{k\in[n_{\mathrm{B}}]} \Big(\bigcap_{k=0}^{|q_k|}\Big(\bigcup_{\substack{\ell\in[|\mathcal{W}_k|]\setminus \\ (\{[l_k+1,j_k-1]\} \\ \cup\{l_r : r\in[0,|q_k|]\})}} \mathbf{DB}(\mathsf{w}_\ell)\Big)\Big) \\
&= \mathbf{DB}(q) \cup \bigcup_{k\in[n_{\mathrm{B}}]} \Big(\bigcap_{k=0}^{|q_k|}\Big(\bigcup_{\substack{\ell\in[|\mathcal{W}_k|]\setminus \\ (\{[l_k+1,j_k-1]\} \\ \cup\{l_r : r\in[0,|q_k|]\})}} \mathbf{DB}(\mathsf{w}_\ell)\Big)\Big)
\end{aligned}
$$

Clearly, from the above expression $\mathbf{DB}(q) \subseteq \mathbf{DB}(q_{mkw})$, which proves the correctness of results for TWINSSE.

4 Detailed Analysis and Discussion on the Leakage of TWINSSE$_{\mathrm{OXT}}$

In this section, we first detail the leakage profile for the specific instantiation of TWINSSE based on the OXT scheme, namely TWINSSE$_{\mathrm{OXT}}$. We then present a discussion on the leakage profiles for both TWINSSE and TWINSSE$_{\mathrm{OXT}}$.

4.1 Leakage Profile of TWINSSE$_{\mathrm{OXT}}$

In this section, we describe the leakage profile of TWINSSE$_{\mathrm{OXT}}$. We begin by recalling from [1] the leakage profile of the original OXT scheme. We then build upon it to describe the leakage profile of TWINSSE$_{\mathrm{OXT}}$, which is actually very similar in spirit to the leakage profile of OXT.

Setup Leakage. The setup leakage in the OXT scheme consists of the size of the database **DB**, which is nothing but the total number of keyword-document pairs in the database **DB**, formally defined as

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

where $\Delta = \{\mathsf{w}_1, \ldots, \mathsf{w}_N\}$ is the dictionary over which the database **DB** is defined.

Search Leakages. Next, we summarize the leakages incurred by OXT during conjunctive keyword search queries.

Result Pattern Leakage: The server learns the final set of document identifiers matching the query. Formally, for a conjunctive query q, the result pattern leakage RP is defined as

$$\mathsf{RP}(q) = \mathbf{DB}(q).$$

Size Pattern Leakage. The server learns the frequency of the s-term (where s-term again refers to the least frequent keyword in the conjunction). Formally, for a conjunctive query $q = (\mathsf{w}_1 \wedge \ldots \wedge \mathsf{w}_n)$, where w_1 is the least frequent keyword in the conjunction, the size pattern SP is defined as

$$\mathsf{SP}(q) = |\mathbf{DB}(\mathsf{w}_1)|.$$

Equality Pattern Leakage. The server learns if two (or more) conjunctive queries have the same s-term (where s-term again refers to the least frequent keyword in the conjunction). Formally, for a sequence of conjunctive queries $(q_1, \ldots, q_M)$, where for each $i \in [M]$, we have

$$q_i = (\mathsf{w}_{i,1} \wedge \ldots \wedge \mathsf{w}_{i,n_i}),$$

where $\mathsf{w}_{i,1}$ is the least frequent keyword in the conjunction, the equality pattern leakage EP is defined as an $M \times M$ matrix where for each $i, j \in [M]$, we have

$$\mathsf{EP}[i,j] = \begin{cases} 1 & \text{if } \mathsf{w}_{i,1} = \mathsf{w}_{j,1}, \\ 0 & \text{otherwise.} \end{cases}$$

Conditional Intersection Pattern Leakage. The server learns if two (or more) conjunctive queries have one or more x-terms in common (where x-term refers any keyword other than the least frequent keyword in the conjunction); more concretely, if two (or more) conjunctive queries have one or more x-terms in common, then the server learns the intersection of the set of documents containing the corresponding s-terms. Formally, for a sequence of conjunctive queries $(q_1, \ldots, q_M)$, where for each $i \in [M]$, we have

$$q_i = (\mathsf{w}_{i,1} \wedge \ldots \wedge \mathsf{w}_{i,n_i}),$$

where $\mathsf{w}_{i,1}$ is the least frequent keyword in the conjunction, the conditional intersection pattern leakage IP is defined as an $M \times M$ matrix of lists, where for each $i, j \in [M]$, we have

$$\mathsf{IP}[i,j] = \begin{cases} \mathbf{DB}(\mathsf{w}_{i,1}) \cap \mathbf{DB}(\mathsf{w}_{j,1}) & \text{if } \overline{\mathsf{IP}}[i,j] = 1, \\ \phi & \text{if } \overline{\mathsf{IP}}[i,j] = 0, \end{cases}$$

where $\overline{\mathsf{IP}}[i,j] = 1$ if and only if there exists at least one pair $(\ell_i, \ell_j) \in [n_i] \times [n_j]$ such that $\mathsf{w}_{i,\ell_i} = \mathsf{w}_{j,\ell_j}$; otherwise, we have $\overline{\mathsf{IP}}[i,j] = 0$.

Security of $\text{TWINSSE}_{\text{OXT}}$**.** We now formalize the security of $\text{TWINSSE}_{\text{OXT}}$ in terms of the leakage profiles described above. We do this using a formal theorem, which may be viewed as a specialization of Theorem 4.2 (in the main paper) to a specific instantiation of TWINSSE based on OXT. Once again, this theorem is based on the (adaptive) simulation-security definition of SSE in the real world-ideal world paradigm, described in Appendix 1.

Theorem 4.1 (Security of $\text{TWINSSE}_{\text{OXT}}$)**.** $\text{TWINSSE}_{\text{OXT}}$ *is an (adaptively) secure SSE scheme with respect to the leakage function* $\mathcal{L}_{\text{TWINSSE}_{\text{OXT}}} = (\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}_{\text{OXT}}}, \mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\text{OXT}}})$*, where for any plaintext database* **DB***, any sequence of conjunctive queries* $\mathcal{Q}_0 = (q_{1,0}, \ldots, q_{M,0})$ *and any sequence of disjunctive queries* $\mathcal{Q}_1 = (q_{1,1}, \ldots, q_{M',1})$*, and any pair of bucketization parameters* (n', n_{B})*, we have*

$$\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}_{\text{OXT}}}(\mathbf{DB}) = (|\widehat{\mathbf{DB}}|, n', n_{\text{B}}),$$

where $\widehat{\mathbf{DB}} = \textsc{GenMetaDB}(\mathbf{DB}, n', n_{\text{B}})$*, and*

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\text{OXT}}}(\mathcal{Q}_0, \mathcal{Q}_1) = [\mathsf{RP}, \mathsf{SP}, \mathsf{EP}, \mathsf{IP}](\mathcal{Q}_0, \mathcal{Q}_{\mathsf{mkw},1}),$$

where $\mathcal{Q}_{\mathsf{mkw},1}$ *is a sequence of (sub-)meta-queries of the form*

$$\mathcal{Q}_{\mathsf{mkw},1} = \{q_{\mathsf{mkw},k,\ell}\}_{k\in[n_{\text{B}}],\ell\in[M']},$$

where for each $\ell \in [M']$*, we have*

$$q_{\mathsf{mkw},\ell} = \left(\bigvee_{k\in[n_{\text{B}}]} q_{\mathsf{mkw},k,\ell}\right) = \textsc{GenMQuery}(q_{\ell,1}, n', n_{\text{B}}).$$

Proof. We defer the formal proof of this theorem to Appendix 5.

4.2 Discussion on the Leakage Profile of $\text{TWINSSE}_{\text{OXT}}$

In this subsection, we present a more in-depth analysis of the leakage profile for $\text{TWINSSE}_{\text{OXT}}$ during conjunctive and disjunctive search queries, and its implications.

Output Leakage. We begin by noting that the output leakage (alternatively, the result pattern leakage) is incurred by nearly all existing SSE schemes, including static and dynamic schemes, in the setting of both single and conjunctive keyword searches (such as in [2, 1, 3, 4, 5, 6]). This is usually considered acceptable in the SSE literature; indeed the few known data/query recovery attacks that manage to exploit this leakage ([7, 8, 9, 10]) assume extremely strong adversarial models where the adversary has partial knowledge of the plaintext database/queries.

s**-Term Leakages.** We focus next on the leakages related to the s-term, namely the size and equality pattern leakages. We begin by noting that these leakages

are somewhat inherent to the design paradigm of OXT, which we base our instantiation of TWINSSE on. Even in the simpler setting of single keyword search, most existing schemes [2, 11, 12, 13, 4, 5, 6] also incur size and equality pattern leakages; the only constructions not to incur such leakages seem to rely on the use of ORAM-style data structures [4, 5]. Fortifying $\text{TWINSSE}_{\text{OXT}}$ with such data structures in an attempt to prevent this leakage is an interesting open challenge, although this would probably have to trade-off with some degradation in search performance (mostly in terms of communication complexity and number of rounds of communication during searches).

It is also possible (and perhaps conceptually simpler) to mask this leakage by using volume-hiding techniques such as padding and encrypted multi-maps (EMMs) [2, 14, 15, 16, 17]. This would incur a degradation in search performance, and it is up to the designer to decide on a suitable trade-off between performance and leakage.

However, we would like to point out that there are no known data/query recovery attacks on SSE schemes that specially exploit leakages related to the s-term. So we believe that *even without the aforementioned fortifications*, it appears that our $\text{TWINSSE}_{\text{OXT}}$ scheme is not vulnerable to any known attacks due to the leakages related to the s-term, in realistic/practical adversarial settings.

x-Term Leakages. Next, we focus on the x-term leakages. We again note that these leakages are somewhat inherent to the design paradigm of OXT, which we base our instantiation of TWINSSE on.. The only known attack on conjunctive SSE schemes that exploits a form of x-term leakages is the *file injection attack* proposed by Zhang *et al.* in [9]. More concretely, the adversarial server must be able to compute $|\mathbf{DB}(\mathsf{w}_1) \cap \mathbf{DB}(\mathsf{w}_i)|$ when processing the search query.

We note however that for file injection attacks to work efficiently, the adversarial server must recover, for every x-term w_i, the result size corresponding to each sub-query of the form $\mathsf{w}_1 \cap \mathsf{w}_i$. However, the x-term leakage profile of $\text{TWINSSE}_{\text{OXT}}$ is not sufficient to compute this term, since the set of xtoken values sent to the server is randomly permuted inside the underlying OXT instantiation precisely to mask such inference-style attacks. Further, one could also instantiate our generic construction of TWINSSE from other conjunctive SSE schemes such as HXT [3] that improve upon OXT in terms of provable security against leakage-based cryptanalysis and file-injection attacks.

Finally, fortifying implementations of $\text{TWINSSE}_{\text{OXT}}$ by using either ORAM-style data structures or adopting volume-hiding techniques such as padding/EMMs may be useful in masking this leakage even further; however, even without such additional fortifications, it appears that our $\text{TWINSSE}_{\text{OXT}}$ scheme is not vulnerable to file injection attacks, or any other known attacks for that matter, due to the leakages related to the s-term, in realistic/practical adversarial settings.

Leakage Cryptanalysis. Looking ahead, in Appendix 6, we present a leakage-based cryptanalysis of the $\text{TWINSSE}_{\text{OXT}}$ scheme via experiments over the Enron email corpus. Our experiments help substantiate that the leakages incurred by the disjunctive search protocol in $\text{TWINSSE}_{\text{OXT}}$ are reasonably benign in practice

and are quite resistant to even the most powerful leakage-based cryptanalysis techniques in the SSE literature over real-world databases, such as those in [8, 9]. We leave it as an open question to extend the analysis using the more advanced leakage cryptanalysis techniques, such as those proposed in [10, 18].

5 Security Proofs for TWINSSE and $\text{TWINSSE}_{\text{OXT}}$

In this section, we formally prove the security of TWINSSE and $\text{TWINSSE}_{\text{OXT}}$ with respect to the generic and specific leakage profiles described in Theorems 4.2 (in the main paper) and 4.1, respectively.

5.1 Proof of Theorem 4.2 (Security Analysis of TWINSSE)

We provide a simulation-based proof approach for TWINSSE. We assumed that the underlying adaptively secure CSSE has the following leakage profile.

$$\mathcal{L}_{\text{CSSE}} = (\mathcal{L}^{\text{Setup}}_{\text{CSSE}}, \mathcal{L}^{\text{Search}}_{\text{CSSE}})$$

We express the leakage of TWINSSE as,

$$\mathcal{L}_{\text{TWINSSE}} = (\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}}, \mathcal{L}^{\text{Search}}_{\text{TWINSSE}})$$

where,

$$\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}}(\mathbf{DB}) = \mathcal{L}^{\text{Setup}}_{\text{CSSE}}(\widehat{\mathbf{DB}})$$

and, $\widehat{\mathbf{DB}} = \text{GenMetaDB}(\mathbf{DB}, n', n_{\text{B}})$, and

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}}(q) = \begin{cases} \mathcal{L}^{\text{Search}}_{\text{CSSE}}(q) & \text{if q is conjunctive,} \\ \{\mathcal{L}^{\text{Search}}_{\text{CSSE}}(q_{\mathsf{mkw},k})\}_{k\in[n_{\text{B}}]} & \text{if q is disjunctive,} \end{cases}$$

where

$$q_{\mathsf{mkw}} = \left(\bigvee_{k\in[n_{\text{B}}]} q_{\mathsf{mkw},k} \right) = \text{GenMQuery}(q, n', n_{\text{B}}).$$

We show that TWINSSE is secure against an adaptive semi-honest adversary $\mathcal{A}$, which has access to leakages from TWINSSE. We build a simulator SIM $\widehat{\mathbf{EDB}}$ generation by TWINSSE.Setup, and transcripts for queries over $\widehat{\mathbf{EDB}}$. The simulator simulates the transcripts τ_i for each query q_i. The simulator has the inputs from the leakage function $\mathcal{L}_{\text{TWINSSE}}$ only, with the setup leakage $\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}}$ and the search leakage $\mathcal{L}^{\text{Search}}_{\text{TWINSSE}}$.

Simulating TWINSSE.Setup**:** The following public parameters are available to SIM_{CSSE} as a part of $\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}}$.

$$\{\mathbf{DB}, n', n_{\text{B}}\}$$

The simulator outputs the its version of $\widehat{\mathbf{EDB}}$ according to the simulation process of CSSE (we assumed that CSSE is provably simulation secure).

$$\begin{aligned} ct_{\widehat{\mathbf{EDB}}} &= SIM^{\text{Setup}}_{\text{TWINSSE}}(\mathbf{DB}) \\ &= SIM^{\text{Setup}}_{\text{CSSE}}(\widehat{\mathbf{DB}}) \\ &= SIM^{\text{Setup}}_{\text{CSSE}}(\mathbf{DB}, n', n_B) \end{aligned}$$

Since, CSSE is proven simulation secure, if follows from the simulation security guarantee of CSSE that $ct_{\widehat{\mathbf{EDB}}}$ is indistinguishable from the one generated in the real experiment.

Simulating TWINSSE.Search**:** For conjunctive queries the adversary does not have any advantage from $\mathcal{L}^{\text{Search}}_{\text{TWINSSE}}$ compared to $\mathcal{L}^{\text{Search}}_{\text{CSSE}}$, which exactly same as CSSE. For disjunctive queries we consider the effect of querying using q_{mkw}.

For disjunctive queries, we argue that the adversary $\mathcal{A}$ does not gain any information about the original disjunctive query with this simulation experiment. The distribution of $\widehat{\mathbf{DB}}$ (hence, also for $\widehat{\mathbf{EDB}}$) is abstracted from **DB** by the meta-keywords. The search leakages of CSSE is characterised by the $\mathcal{L}_{\text{CSSE}}$, provided from CSSE construction. Since, CSSE in TWINSSE executes over meta-keyword only, this leakage is expressed in the context of meta-keywords as below.

$$\mathcal{L}'_{\text{CSSE}} = \mathcal{L}_{\text{CSSE}}(meta - keywords)$$

With this leakage information of CSSE, the search leakage of TWINSSE can be expressed as below.

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}}(q) = \mathcal{L}^{\text{Search}}_{\text{TWINSSE}}(q_{\mathsf{mkw},k})_{k\in[n_{\text{B}}]} = \{\mathcal{L}'_{\text{CSSE}}, n_{\text{B}}, n'\}$$

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

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}}(q) = \mathcal{L}^{\text{Search}}_{\text{TWINSSE}}(q_{\mathsf{mkw},k})_{k\in[n_{\text{B}}]} = \{\mathcal{L}'_{\text{CSSE}}\}$$

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

5.2 Proof of Theorem 4.1 (Security Analysis of $\text{TWINSSE}_{\text{OXT}}$)

We resort to a simulation-based security analysis for $\text{TWINSSE}_{\text{OXT}}$. We assume a semi-honest adversary $\mathcal{A}$ which has access to the leakage from standard SSE leakages in an adaptive model. Security analysis of TWINSSE relies upon the semantic security notions provided by CSSE. TWINSSE inherits these notions through the core OXT (in case of $\text{TWINSSE}_{\text{OXT}}$, the OXT) instance. We assume the following properties of OXT achieves with efficient performance.

1. Primitives used in construction of OXT hold the standard security assumptions.

2. OXT is non-adaptively and adaptively secure with the above assumptions.

We consider the following leakage profile for OXT.

$$\mathcal{L}_{\mathsf{OXT}} = \{\mathcal{L}^{\text{Setup}}_{\mathsf{OXT}}, \mathcal{L}^{\text{Search}}_{\mathsf{OXT}}\}$$

Here, $\mathcal{L}^{\text{Setup}}_{\mathsf{OXT}}$ captures the leakage from the OXT.Setup, and $\mathcal{L}^{\text{Search}}_{\mathsf{OXT}}$ encapsulates the leakage from OXT.Search. More precisely, these can be expressed as,

$$\mathcal{L}^{\text{Setup}}_{\mathsf{OXT}}(\mathbf{DB}) = \{|\mathbf{DB}|\}$$

and

$$\mathcal{L}^{\text{Search}}_{\mathsf{OXT}}(\mathbf{EDB}, \{q_k\}_{q_k \in \mathcal{Q}_0)} = \{RP, SP, EP, IP\}$$

where, $\mathcal{Q}_0$ is a set of conjunctive queries. The leakages RP, SP, EP, and IP are the pattern leakages from OXT (see Appendix 4.2).

We define the leakage profile of TWINSSE$_{\mathsf{OXT}}$ with respect to these above definitions and assumptions as below.

$$\mathcal{L}_{\text{TWINSSE}_{\mathsf{OXT}}} = \{\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}_{\mathsf{OXT}}}, \mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\mathsf{OXT}}}\}$$

The leakage functions above can be expressed as

$$\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}_{\mathsf{OXT}}}(\mathbf{DB}) = \{|\widehat{\mathbf{DB}}|, n', n_{\mathsf{B}}\}$$

and

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\mathsf{OXT}}}(\widehat{\mathbf{EDB}}, \mathcal{Q}_0, \mathcal{Q}_1) = [\mathsf{RP}, \mathsf{SP}, \mathsf{EP}, \mathsf{IP}](\mathcal{Q}_0, \mathcal{Q}_{\mathsf{mkw},1}),$$

.

For conjunctive queries,

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\mathsf{OXT}}}(\widehat{\mathbf{EDB}}, \{q_k\}_{q_k \in \mathcal{Q}_0}) = [\widehat{\mathsf{RP}}, \widehat{\mathsf{SP}}, \widehat{\mathsf{EP}}, \widehat{\mathsf{IP}}]$$

.

Here, $\{\widehat{\mathsf{RP}}, \widehat{\mathsf{SP}}, \widehat{\mathsf{EP}}, \widehat{\mathsf{IP}}\}$ are the $\{\mathsf{RP}, \mathsf{SP}, \mathsf{EP}, \mathsf{IP}\}$ leakages in the context of meta-keywords. For conjunctive queries, it is exactly the same as OXT.

Since, OXT is simulation secure against these leakages, simulation security of TWINSSE$_{\mathsf{OXT}}$ for conjunctive queries is straightforwardly implied from OXT.

In disjunctive queries, the query transformation process is carried out locally by the client, and the actual search is completed using OXT.Search protocol, we can write TWINSSE$_{\mathsf{OXT}}$.Search leakage as

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\mathsf{OXT}}}(\widehat{\mathbf{EDB}}, \{q_{mkw,1,k}\}_{k \in [\mathcal{Q}_1]}) = \{\widehat{\mathsf{RP}}, \widehat{\mathsf{SP}}, \widehat{\mathsf{EP}}, \widehat{\mathsf{IP}}\}$$

.

We build a simulator SIM to simulate the $\widehat{\mathbf{EDB}}$ generation by $\text{TWINSSE}_{\text{OXT}}$ from **DB**, and transcripts for query search over $\widehat{\mathbf{EDB}}$. The simulator simulates the transcripts τ_i for each query $q_i \in \mathcal{Q}$. The simulator has the inputs from the leakage function $\mathcal{L}_{\text{TWINSSE}_{\text{OXT}}}$ only, with the setup leakage $\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}_{\text{OXT}}}$ and the search leakage $\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\text{OXT}}}$.

Simulating Setup: The following public parameters are available to SIM_{OXT} as a part of $\mathcal{L}^{\text{Setup}}_{\text{TWINSSE}_{\text{OXT}}}$.

$$\{|\mathbf{EDB}|, |\widehat{\Delta}|\}$$

The simulator outputs the its version of $\widehat{\mathbf{EDB}}$ according to the simulation process of OXT (we assumed that OXT is provably simulation secure).

$$ct_{\widehat{\mathbf{EDB}}} = SIM_{\mathsf{OXT}}.\textsc{Setup}(|\mathbf{MDB}|, |\widehat{\Delta}|)$$

It follows from the simulation security guarantee of OXT that $\mathsf{ct}_{\widehat{\mathbf{EDB}}}$ is indistinguishable from the one generated in the real experiment.

Simulating Search: For the conjunctive queries, the leakage $\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\text{OXT}}}$ is exactly the same as $\mathcal{L}^{\text{Search}}_{\text{OXT}}$. Hence, we can write the following.

$$\mathcal{L}^{\text{Search}}_{\text{TWINSSE}_{\text{OXT}}}(\widehat{\mathbf{EDB}}, \{q_k\}_{k\in[|\mathcal{Q}|]}) = \mathcal{L}^{\text{Search}}_{\text{OXT}}(\mathbf{EDB}, \{q_k\}_{k\in[|\mathcal{Q}|]})$$

By the simulation security guarantee of OXT, $\text{TWINSSE}_{\text{OXT}}$ secure against these leakages.

For disjunctive queries, we argue that the adversary $\mathcal{A}$ does not gain any information about the original disjunctive query except $|q|$. The distribution of **MDB** (encrypted to $\widehat{\mathbf{EDB}}$) is abstracted from **DB** through the meta-keywords. We resort to a more conservative analysis for this proof, as keywords do not have direct inference from meta-keywords, especially that is applicable over any database in general. The position of each w in an mkw is fixed according to the frequency of w, which is unique for a **DB**. The lemmas below relate worst cases where an inference can be established between the query keywords and the corresponding meta-keywords without any additional knowledge of the plain database.

Lemma 5.1, Lemma 5.2, and Lemma 5.3 relates the disjunctive q with $\mathsf{w}_i \in \Delta$ to the conjunctive q with $\mathsf{mkw}_i \in \widehat{\Delta}$.

Lemma 5.1. *Consider two disjunctive queries of the same length t*

$$q_0 = \mathsf{w}_{1,q_0} \vee \mathsf{w}_{2,q_0} \vee \ldots \vee \mathsf{w}_{t,q_0},\ \mathsf{w}_{i,q_0}$$
$$q_1 = \mathsf{w}_{1,q_1} \vee \mathsf{w}_{2,q_1} \vee \ldots \vee \mathsf{w}_{t,q_1},\ \mathsf{w}_{i,q_1}$$

have the following expressions using mkw*s,*

$$q_0 = q_{0,\mathsf{mkw}} = \mathsf{mkw}_{1,q_0} \wedge \mathsf{mkw}_{2,q_0} \wedge \ldots \wedge \mathsf{mkw}_{t+1,q_0}$$
$$q_1 = q_{1,\mathsf{mkw}} = \mathsf{mkw}_{1,q_1} \wedge \mathsf{mkw}_{2,q_1} \wedge \ldots \wedge \mathsf{mkw}_{t+1,q_1}$$

both of length $t+1$, and the mkw*s are placed in the increasing order of the starting index of the* 0*s stretch in each* mkw*. If the* mkw*s at index k in q_0 and q_1 are the same, then* $\mathsf{w}_{k-1,q_0} = \mathsf{w}_{k-1,q_1}$ *and* $\mathsf{w}_{k,q_0} = \mathsf{w}_{k,q_1}$.

Proof. The proof of Lemma 5.1 is given in Section 5.3.1. □

Lemma 5.2. *Consider two disjunctive queries q_0 and q_1, of the same length t have the* mkw *expressions as defined in Lemma 5.1 - both of length $t+1$. If the* mkw*s at indices k_0 in q_0, and k_1 in q_1 are the same, then* $\mathbf{x}_{k_0-1,q_0} = \mathsf{w}_{k_1-1,q_1}$ *and* $\mathsf{w}_{k_0,q_0} = \mathsf{w}_{k_1,q_1}$.

Proof. The proof of Lemma 5.2 is given in Section 5.3.2. □

Lemma 5.3. *Consider two disjunctive queries of different length t_0 and t_1 -*

$$q_0 = \mathsf{w}_{1,q_0} \vee \mathsf{w}_{2,q_0} \vee \ldots \vee \mathsf{w}_{t_0,q_0},\ \mathsf{w}_{i,q_0} \in \Delta$$
$$q_1 = \mathsf{w}_{1,q_1} \vee \mathsf{w}_{2,q_1} \vee \ldots \vee \mathsf{w}_{t_1,q_1},\ \mathsf{w}_{i,q_1} \in \Delta$$

have following expressions in the mkw*s*

$$q_0 = q_{0,\mathsf{mkw}} = \mathsf{mkw}_{1,q_0} \wedge \mathsf{mkw}_{2,q_0} \wedge \ldots \wedge \mathsf{mkw}_{t_0+1,q_0}$$
$$q_1 = q_{1,\mathsf{mkw}} = \mathsf{mkw}_{1,q_1} \wedge \mathsf{mkw}_{2,q_1} \wedge \ldots \wedge \mathsf{mkw}_{t_1+1,q_1}$$

which are of lengths t_0+1 and t_1+1 respectively. If the mkw*s at indices k_0 in q_0, and k_1 in q_1 are the same, then* $\mathsf{w}_{k_0-1,q_0} = \mathsf{w}_{k_1-1,q_1}$ *and* $\mathsf{w}_{k_0,q_0} = \mathsf{w}_{k_1,q_1}$.

Proof. The proof of Lemma 5.3 is given in Section 5.3.3. □

Recall that, the query transformation is executed by the client locally. The search is executed as a two-party protocol between the client and the server using the meta-keywords. The server learns $|q|$ trivially from q_{mkw} through of meta-keywords. From Lemma 5.1, 5.2, and 5.3, an adversary can infer the position of the same ws in two queries of same length or different lengths if both queries have a *common* mkw in them.

However, the server can only infer if the least-frequent mkws in q_{mkw} are identical or not in mkw expressions of two qs from $\widehat{SP}$. The mkw expressions in each of the three lemmas require to place mkws in increasing order of the starting index of the 0's stretch. Whereas, the actual query expression for OXT has the least-frequent mkw first. No direct inference can be conjectured for the least-frequent mkw and the query expressions in the lemmas. Hence, an adversary $\mathcal{A}$ can not distinguish between the common meta-keyword and a distinct meta-keyword.

In the case, where the least-frequent of mkws is the first one in the query expression of the lemmas too, the first keyword is also the same for both ws. This is equivalent to the case of two conjunctive queries in keywords having the least-frequent w same.

Therefore, the leakage from $\text{TWINSSE}_{\text{SEARCH}}$ can be limited to the OXT pattern leakages only, as expressed below.

$$\mathcal{L}^{\text{SEARCH}}_{\text{TWINSSE}_{\text{OXT}}}(\widehat{\mathbf{EDB}}, \{q_k\}_{k\in[|\mathcal{Q}|]}) = \{\mathcal{L}'_{\text{OXT}}, |q_k|_{k\in[|\mathcal{Q}|]}\}$$

Since, OXT is proven simulation secure, if follows from the simulation security guarantee that $\mathcal{A}$ no additional advantage over the real experiment.

5.3 Proofs of the Lemmas

We present the proofs of the lemmas presented earlier in this section. We follow the notations and conventions as used in the main body of the paper.

5.3.1 Proof of Lemma 5.1

Proof. By construction, each meta-keyword mkw_i has the original keywords appearing in sorted order in the binary string representation (increasing order of frequency from left to right). Assume, the k'th meta-keyword mkw_k is same for both the queries q_0 and q_1. Without loss of generality, a meta-keyword in the basic $O(N^2)$ ($\text{TWINSSE}_{\text{BASIC}}$) method can be formed as

$$\{b_1, b_2, \ldots, b_r, b_{r+1}, \ldots, b_s, b_{s+1}, \ldots, b_n\},\ b_i \in \{0, 1\}$$

where $1 \leq r < s \leq n$, and $b_i = 0$ for $r < i < s$.

To have an mkw of this form, q must have two keywords at indices r and s, and none in between (for q_0 and q_1 both). Since the mkws are constructed using ws in sorted order, if both queries q_0 and q_1 have the same r and same s (as one mkw is the same), the keywords w_r and w_s in both q_0 and q_1 are also the same. Hence, we have $\mathsf{w}_{k-1,q_0} = \mathsf{w}_{k-1,q_1}$ and $\mathsf{w}_{k,q_0} = \mathsf{w}_{k,q_1}$. □

5.3.2 Proof of Lemma 5.2

Proof. We assume the common mkw of q_0 and q_1 can be expressed as

$$\{b_1, b_2, \ldots, b_r, b_{r+1}, \ldots, b_s, b_{s+1}, \ldots, b_n\},\ b_i \in \{0, 1\}$$

where $1 \leq r < s \leq n$, and $b_i = 0$ for $r < i < s$. The mkw appears at indices k_0 in q_0 and at k_1 in q_1. Since the indices of ws in the mkw strings are in sorted order (increasing frequency) and remains fixed for all mkws, the ws at index r and index s are the same for both q_0 and q_1. However, as the index of mkw is different in q_0 and q_1, the number of preceding ws before index r in q_0 and q_1 are different, equal to $k_0 - 2$ and $k_1 - 2$ respectively. Hence, for q_0, r is equal to $k_0 - 1$, and equal to $k_1 - 1$ in q_1. Following the above argument, we have $\mathsf{w}_{k_0-1,q_0} = \mathsf{w}_{k_1-1,q_1}$ and $\mathsf{w}_{k_0,q_0} = \mathsf{w}_{k_1,q_1}$. □

5.3.3 Proof of Lemma 5.3

Proof. The proof of Lemma 5.3 follows from the proof of Lemma 5.2. Essentially, Lemma 5.3 is the extension of Lemma 5.2 for two different lengths of queries. Intuitively, it can be established in the following way. Recall that in Proof 5.3.2, r and s remains same in both q_0 and q_1, as in binary representation all mkws and qs have the same length n. However, the number of ws in q changes, and consequently, number of mkws change. Hence, the range of indices k_0 and k_1 are different for q_0 and q_1. This does not affect r and s which are positions of keywords (not related to number of keywords) in the binary representation of fixed length. Hence, the same argument from the proof of Lemma 5.2 holds. □

6 Leakage Cryptanalysis of TWINSSE$_{\text{OXT}}$

In analyzing the leakage profile for TWINSSE$_{\text{OXT}}$, we followed a two-step process:

1. **Step 1:** Provably establishing the leakage using a simulation-based framework (see Appendix 5).
2. **Step-2:** Analyzing its impact with respect to well-known attacks such as leakage-abuse and file-injection attacks (please see Appendix 4 for a detailed discussion on why the leakage profile of TWINSSE$_{\text{OXT}}$ is highly resistant to a large class of existing attacks).

Unfortunately, the SSE literature does not currently have well-defined metrics for *experimentally* evaluating the leakage of a scheme over real-world databases. This is why the two-step approach of leakage enumeration and analysis as mentioned above is accepted widely as the norm in the SSE literature (notably [2, 1, 12, 13, 4, 3, 5].

In this section, we encapsulate Step-2 of our leakage profile analysis for TWINSSE$_{\text{OXT}}$ into the following claim:

Claim 6.1 (Informal)**.** *Conjunctive and disjunctive search operations in* TWINSSE$_{\text{OXT}}$ *incur leakages that are almost identical to those incurred by* OXT *[1] and* HXT *[3], and are reasonably benign in practice based on analysis already done in prior works.*

We substantiate this claim by leakage cryptanalysis experiments on conjunctive and disjunctive keyword searches in TWINSSE$_{\text{OXT}}$. The experiments were conducted over the same Enron email corpus as was used for the performance evaluation experiments in the main paper. Throughout, we use a bucket size $n' = 10$ (same as for the performance evaluation experiments in Section 5 in main paper.

6.1 Leakage Cryptanalysis of Conjunctive Search Queries in TWINSSE$_{\text{OXT}}$

We now evaluate the leakage from the conjunctive search protocol in TWINSSE$_{\text{OXT}}$. We evaluate the probability that the adversary guesses correctly the keywords w_1

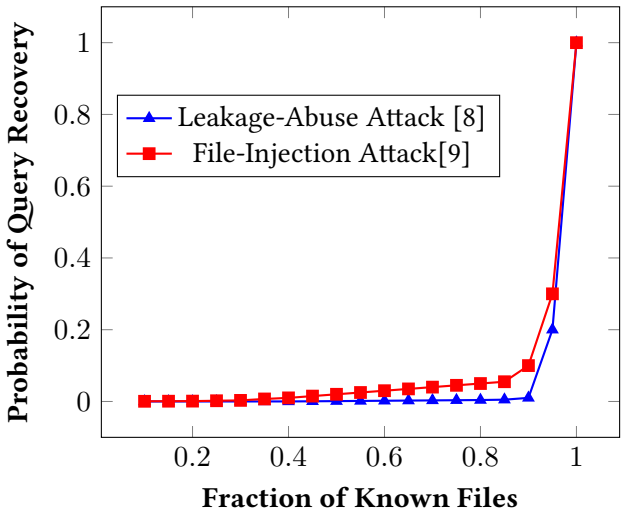

Figure 1: Leakage Analysis of TWINSSE$_{\text{OXT}}$: Two-Keyword Conjunctive Searches in the "Known Files" Setting

and w_2 underlying a two-conjunction query $q = (w_1 \wedge w_2)$ by one of two well-known and extensively studied cryptanalysis methodologies in the SSE literature-the *leakage-abuse attack* of Cash *et al.* [8] and the *file-injection attack* of Zhang *et al.* [9]. These attacks operate in two models - the *known* file model (where the adversary knows the contents of a certain fraction of the files in the database) and the *chosen/injected* file model (where a certain fraction of the files in the database are adversarially generated).

Naturally, when the adversary knows (or has injected) all the documents in the database, query recovery is trivial. However, this is a very strong attack model and is practically infeasible. What we want in a real-life application is that when the adversary knows only a small fraction of the files in the database, or has managed to inject a small fraction of files into the database, query recovery should happen with a very small probability. This would essentially indicate that the adversary has access to no additional leakage (about either the keywords underlying the query or the files in the database) from the search protocol beyond the benign leakage profile that was formally enumerated in Appendix 5.

Figure 1 illustrates the success probability of the adversary for both kinds of attacks in the "known file" attack setting. The results clearly establish that even when the fraction of known files in the database is as high as 50%, the success probability of the adversary in recovering the keywords underlying a conjunction $(w_1 \wedge w_2)$ is less than 5%.

Similarly, Figure 1 illustrates the success probability of the adversary for both kinds of attacks in the "chosen/injected file" attack setting. The results again establish that even when the fraction of injected files in the database is as high as 60% (which is quite unlikely in any real world database), the success probability

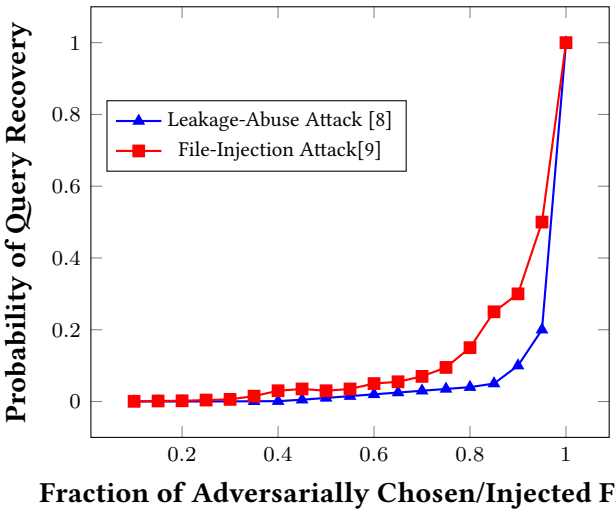

Figure 2: Leakage Analysis of TWINSSE$_{\text{OXT}}$: Two-Keyword Conjunctive Searches in the "Chosen Files" Setting

of the adversary in recovering the keywords underlying a conjunction $(\mathsf{w}_1 \wedge \mathsf{w}_2)$ is less than 5%.

Our experiments thus substantiate our statement in Claim 6.1 that the leakages incurred by the disjunctive search protocol in TWINSSE$_{\text{OXT}}$ are reasonably benign in practice and are quite resistant to even the most powerful leakage-based cryptanalysis techniques in the SSE literature over real-world databases.

6.2 Leakage Cryptanalysis of Disjunctive Search Queries in TWINSSE$_{\text{OXT}}$

We now evaluate the leakage from the disjunctive search protocol in TWINSSE$_{\text{OXT}}$. We evaluate the probability that the adversary guesses correctly the keywords w_1 and w_2 underlying a two-disjunction query $q = (\mathsf{w}_1 \vee \mathsf{w}_2)$ via the same attack methods and the same attack settings in the previous subsection.

Figure 3 illustrates the success probability of the adversary for both kinds of attacks in the "known file" attack setting. The results clearly establish that even when the fraction of known files in the database is as high as 50%, the success probability of the adversary in recovering the keywords underlying a disjunction $(\mathsf{w}_1 \vee \mathsf{w}_2)$ is less than 5%.

Similarly, figure 3 illustrates the success probability of the adversary for both kinds of attacks in the "chosen/injected file" attack setting. The results again establish that even when the fraction of injected files in the database is as high as 60% (which is quite unlikely in any real world database), the success probability of the adversary in recovering the keywords underlying a disjunction $(\mathsf{w}_1 \vee \mathsf{w}_2)$ is less than 5%.

Our experiments additionally substantiate our statement in Claim 6.1 that the leakages incurred by the disjunctive search protocol in TWINSSE$_{\text{OXT}}$ are reason-

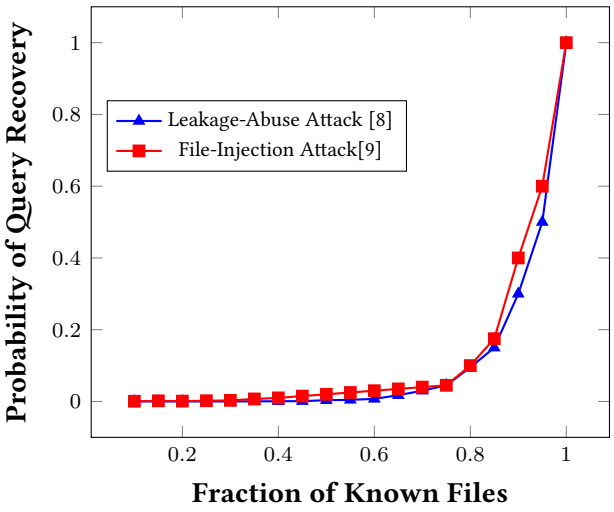

Figure 3: Leakage Analysis of TWINSSE$_{\text{OXT}}$: Two-Keyword Disjunctive Searches in the "Known Files" Setting

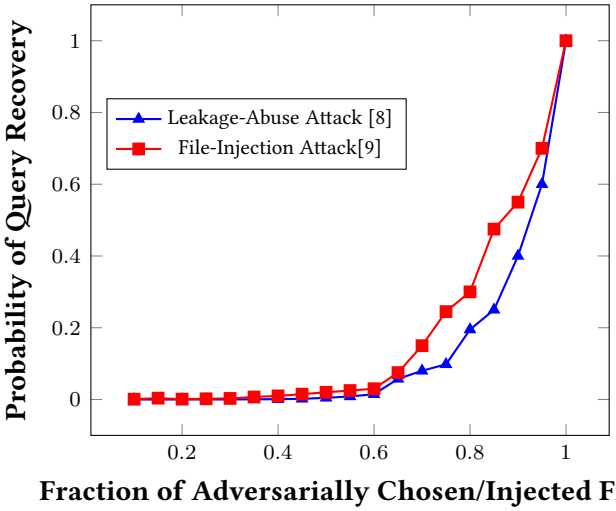

Figure 4: Leakage Analysis of TWINSSE$_{\text{OXT}}$: Two-Keyword Disjunctive Searches in the "Chosen Files" Setting

ably benign in practice and are quite resistant to even the most powerful leakage-based cryptanalysis techniques in the SSE literature over real-world databases.

Finally, we leave it as an open question to extend the above analysis using the more advanced leakage cryptanalysis techniques, such as those proposed in [10, 18].

7 General Boolean Queries (CNF and DNF) in TWINSSE and $\text{TWINSSE}_{\text{OXT}}$

In Section 4.3 (in main paper), we described how TWINSSE and its instantiation from OXT, namely $\text{TWINSSE}_{\text{OXT}}$, handle purely conjunctive and purely disjunctive queries. In this section, we describe how TWINSSE can be extended to address general Boolean queries in either the conjunctive normal form (CNF) or the disjunctive normal form (DNF) form.

We note here that OXT does support Boolean queries beyond simple conjunctions, albeit where the query must be in a restricted *searchable normal form* (SNF) [1]; our transformation is significantly more general in the sense that it extends to any CNF or DNF formula over keywords, well beyond the scope of SNF queries.

We begin by describing how to handle DNF queries, because, similar to its purely conjunctive and disjunctive counterparts, DNF queries are also handled by TWINSSE (and hence, by extension, $\text{TWINSSE}_{\text{OXT}}$) by making fully black-box usage of the underlying conjunctive SSE scheme. Subsequently, we show how to address CNF queries. This is slightly more involved, and makes non black-box usage of the underlying conjunctive SSE scheme (we describe a specific strategy for $\text{TWINSSE}_{\text{OXT}}$ to handle CNF queries that relies on a special data structure used by the OXT scheme).

7.1 Handling Boolean Queries in DNF Form

In Boolean logic, a disjunctive normal form (DNF) is a canonical normal form of a logical formula consisting of a disjunction of conjunctions (alternatively, OR of AND clauses). Formally, any query q that is a Boolean formula over keywords in the DNF form takes the form

$$q = \bigvee_{\ell \in [L]} q_\ell = \bigvee_{\ell \in [L]} (\mathsf{w}_{\ell,1} \wedge \ldots \wedge \mathsf{w}_{\ell,t_\ell}),$$

where each $q_\ell = (\mathsf{w}_{\ell,1} \wedge \ldots \wedge \mathsf{w}_{\ell,t_\ell})$ for $\ell \in [L]$ is a conjunctive *clause*. Our approach to handle a DNF query is straightforward, and closely resembles, at a high level, our strategy for handling disjunctive queries via query partitioning in TWINSSE. Let $\text{CSSE} = (\text{CSSE.}\textsc{Setup}, \text{CSSE.}\textsc{Search})$ be any generic conjunctive SSE scheme. The search algorithm processes q via the following steps (the setup algorithm remains the same as TWINSSE.Setup described in Algorithm 1 (main paper):

- **Client:** Parse a DNF query as $q = \bigvee_{\ell \in [L]} q_\ell$.
- **Client + Server:** For each $\ell \in [L]$ (either in parallel or in uniformly random order), compute
  $\mathbf{DB}(q_\ell) = \text{CSSE.}\textsc{Search}(q_\ell, \widehat{\mathbf{EDB}})$,
  where $\widehat{\mathbf{EDB}}$ is the encrypted meta-database output by TWINSSE.Setup.

- **Client:** Locally compute at the client

$$\mathbf{DB}(q) = \bigcup_{\ell \in [L]} \mathbf{DB}(q_\ell).$$

Correctness. Correctness of search follows immediately from the correctness guarantees of the underlying conjunctive SSE scheme CSSE.

Search Complexity. We present an (asymptotic) analysis of the complexity of handling DNF search queries (more concretely, the computational and communication requirements during DNF query processing) when we instantiate TWINSSE using the OXT protocol from [1], i.e., in $\text{TWINSSE}_{\text{OXT}}$. Let q be a DNF query of the form

$$q = \bigvee_{\ell \in [L]} q_\ell = \bigvee_{\ell \in [L]} (\mathsf{w}_{\ell,1} \wedge \ldots \wedge \mathsf{w}_{t_\ell,\ell}),$$

where we assume, without loss of generality, that for each $\ell \in [L]$, $\mathsf{w}_{\ell,1}$ is the least frequent conjunct in the conjunctive clause q_ℓ. When processing q using $\text{TWINSSE}_{\text{OXT}}$, the computational costs (at both the client and the server) as well as the communication requirements between the client and the server scale linearly as $O(\gamma_{\text{DNF}})$, where

$$\gamma_{\text{DNF}} = \sum_{\ell \in [L]} t_\ell |\mathbf{DB}(\mathsf{mkw}_{\ell,1})|.$$

Note that this is very similar in flavor to the analysis of disjunctive search query overheads for $\text{TWINSSE}_{\text{OXT}}$ in Section 4 of the main paper.

Leakage Analysis. We state the following theorems for the leakage from TWINSSE and $\text{TWINSSE}_{\text{OXT}}$ when processing Boolean queries in DNF form.

Theorem 7.1 (DNF Query Processing in TWINSSE)**.** *Assuming that* CSSE *is an (adaptively) secure SSE scheme with respect to the leakage function* $\mathcal{L}_{\text{CSSE}} = (\mathcal{L}^{\text{Setup}}_{\text{CSSE}}, \mathcal{L}^{\text{Search}}_{\text{CSSE}})$*, the leakage incurred by* TWINSSE *when processing a DNF query as described above is* $\mathcal{L}^{\text{Search,DNF}}_{\text{TWINSSE}}$*), where for any DNF query* $q = \bigvee_{\ell \in [L]} q_\ell$*, we have*

$$\mathcal{L}^{\text{Search,DNF}}_{\text{TWINSSE}}(q) = \left\{ \mathcal{L}^{\text{Search}}_{\text{CSSE}}(q_\ell) \right\}_{\ell \in [L]}.$$

Theorem 7.2 (DNF Query Processing in $\text{TWINSSE}_{\text{OXT}}$)**.** *The leakage incurred by* $\text{TWINSSE}_{\text{OXT}}$ *when processing a DNF query as described above is* $\mathcal{L}^{\text{Search,DNF}}\text{TWINSSE}_{\text{OXT}}$*, where for any sequence of DNF queries* $\mathcal{Q} = (q_1, \ldots, q_M)$ *such that* $q_m = \bigvee_{\ell \in [L_m]} q_{m,\ell}$ *for each* $m \in [M]$*, we have*

$$\mathcal{L}^{\text{Search,DNF}}_{\text{TWINSSE}_{\text{OXT}}}(\mathcal{Q}) = [\mathsf{RP}, \mathsf{SP}, \mathsf{EP}, \mathsf{IP}](\{\{q_{m,\ell}\}_{\ell \in [L_m]}\}_{m \in [M]}).$$

where RP*,* SP*,* EP *and* IP *leakages for conjunctive queries are as defined in Appendix 4.*

The proofs of these theorems are very similar to the proofs of Theorems 4.2 (in main paper) and 4.1 described earlier in Appendix 5, and are hence not detailed separately.

7.2 Handling Boolean Queries in CNF Form

In Boolean logic, a conjunctive normal form (CNF) is a canonical normal form of a logical formula consisting of a conjunction of disjunctions (alternatively, AND of OR clauses). Formally, any query q that is a Boolean formula over keywords in the CNF form takes the form

$$q = \bigwedge_{\ell \in [L]} q_\ell = \bigwedge_{\ell \in [L]} (\mathsf{w}_{\ell,1} \vee \ldots \vee \mathsf{w}_{\ell,t_\ell}),$$

where each $q_\ell = (\mathsf{w}_{\ell,1} \vee \ldots \vee \mathsf{w}_{\ell,t_\ell})$ for $\ell \in [L]$ is a disjunctive *clause*. Our approach to handle a CNF query is slightly more involved, and makes usage of some specific features of the OXT protocol to ensure sub-linear search overheads in practice. Hence, the subsequent description of how to handle CNF queries is specific to TWINSSE$_{\text{OXT}}$. We leave it as an interesting open question to investigate a generic solution using any conjunctive SSE scheme in a black-box manner.

We now describe our proposed strategy for handling CNF queries in TWINSSE$_{\text{OXT}}$. Before delving into the details, we need to recall some details of the original OXT scheme due to Cash *et al.* [1]. We refer the reader to Appendix 10 for details of the OXT scheme; however, we will try to make the description here as self-contained as possible. The OXT protocol maintains on the server (as part of the encrypted database **EDB**) a special data structure called a "cross-tag set" (or XSet in short). The XSet consists of several "cross-tags", where each cross-tag $\mathsf{xtag}_{\mathsf{id},\mathsf{w}}$ corresponds to a document identity-keyword pair $(\mathsf{id}, \mathsf{w})$, where

$$\mathsf{xtag}_{\mathsf{id},\mathsf{w}} \in \mathsf{XSet} \text{ if and only if } \mathsf{w} \in \mathbf{DB}(\mathsf{id}).$$

In our handling of CNF queries in TWINSSE$_{\text{OXT}}$, we make black-box usage the following sub-functions provided by any implementation of the OXT protocol:

- OXT.GenXTag$(\mathsf{sk}, \mathsf{id}, \mathsf{w})$: The client can use the secret key generated at setup by OXT.Search to generate $\mathsf{xtag}_{\mathsf{id},\mathsf{w}}$ for any document identifier id and keyword w.
- OXT.SearchXTag$(\mathsf{xtag}_{\mathsf{id},\mathsf{w}}; \mathsf{XSet})$: On receipt of a cross-tag $\mathsf{xtag}_{\mathsf{id},\mathsf{w}}$ from the client, the server can look up the XSet efficiently to return a bit $\beta \in \{0, 1\}$, where $\beta = 1$ if $xtag_{\mathsf{id},\mathsf{w}} \in \mathsf{XSet}$, and $\beta = 0$ otherwise.

Given these sub-routines, our proposal for processing a CNF query q proceeds via the steps outlined below (the setup algorithm again remains the same as TWINSSE.Setup described in Algorithm 1 (main paper), albeit for CSSE = OXT). Note that unlike purely conjunctive/disjunctive queries and DNF queries, all of which required a single round search protocol, our processing of CNF queries now requires two rounds of communication between the client and the server.

- **Client:** Parse a CNF query as

$$q = \bigwedge_{\ell \in [L]} q_\ell = \bigwedge_{\ell \in [L]} (\mathsf{w}_{\ell,1} \vee \ldots \vee \mathsf{w}_{\ell,t_\ell}),$$

- **Client:** Identify the candidate disjunctive clause q_ℓ with the smallest result set (this can be computed in a straightforward manner from the client state st output by OXT.$\textsc{Setup}$, which has the frequency of each keyword in the dictionary).

- **Client+Server (Round-1):** Compute the result-set corresponding to the disjunctive clause q_ℓ as

$$\mathbf{DB}(q_\ell) = \text{TWINSSE}_{\text{OXT}}.\textsc{Search}(q_\ell, \widehat{\mathbf{EDB}}),$$

  where $\widehat{\mathbf{EDB}}$ is the encrypted meta-database output by TWINSSE$_{\text{OXT}}$.$\textsc{Setup}$, by directly using the disjunctive search protocol described in Algorithm 3 (main paper) with CSSE = OXT.

- **Client:** For each $\mathsf{id} \in \mathbf{DB}(q_\ell)$ and each $\mathsf{w}_{i,\ell'}$ for $\ell' \neq \ell$ in the query q, compute

$$\mathsf{xtag}_{\mathsf{id},\mathsf{w}_{i,\ell'}} = \text{OXT}.\textsc{GenXTag}(\mathsf{sk}, \mathsf{id}, \mathsf{w}_{i,\ell'}).$$

- **Client+Server (Round-2):** For each $\mathsf{id} \in \mathbf{DB}(q_\ell)$ and each $\mathsf{w}_{i,\ell'}$ (either in parallel or in uniformly random order), the client sends $\mathsf{xtag}_{\mathsf{id},\mathsf{w}_{i,\ell'}}$ to the server and receives in response

$$\beta_{\mathsf{id},\mathsf{w}_{i,\ell'}} = \text{OXT}.\textsc{SearchXTag}(\mathsf{xtag}_{\mathsf{id},\mathsf{w}_{i,\ell'}}; \mathsf{XSet}).$$

- **Client:** For each $\mathsf{id} \in \mathbf{DB}(q_\ell)$, compute

$$\beta_{\mathsf{id}} = \bigwedge_{\ell' \in [L] \setminus \{\ell\}} \left(\beta_{\mathsf{id},\mathsf{w}_{\ell,1}} \vee \ldots \vee \beta_{\mathsf{id},\mathsf{w}_{\ell,t_\ell}} \right).$$

  Output the final result set

$$\mathcal{R}_q = \{\mathsf{id} \in \mathbf{DB}(q_\ell) \text{ such that } \beta_{\mathsf{id}} = 1\}.$$

Correctness. Correctness of search follows immediately from the correctness guarantees of TWINSSE$_{\text{OXT}}$, and the correctness guarantees of the OXT protocol itself.

Search Complexity. We now present an (asymptotic) analysis of the complexity of handling CNF search queries (more concretely, the computational and communication requirements during CNF query processing) in TWINSSE$_{\text{OXT}}$. Let q be a CNF query of the form

$$q = \bigwedge_{\ell \in [L]} q_\ell = \bigwedge_{\ell \in [L]} (\mathsf{w}_{\ell,1} \vee \ldots \vee \mathsf{w}_{\ell,t_\ell}),$$

where we assume, without loss of generality, that $q_1 = (\mathsf{w}_{1,2} \vee \ldots \vee \mathsf{w}_{\ell,2})$ is the disjunctive clause with the smallest result set. Let $q_{\mathsf{mkw}} = \vee_{k \in [n_{\mathsf{B}}]} q_{\mathsf{mkw},k}$ be

the corresponding meta-query when the disjunctive search query corresponding to q_1 is processed using TWINSSE$_{\text{OXT}}$.Search, and assume without loss of generality that $\mathsf{mkw}^{(k)}_{i_k,j_k}$ is the least frequent meta-keyword within $q_{\mathsf{mkw},k}$ for each $k \in [n_{\text{B}}]$ (such that $q_{\mathsf{mkw},k}$ is non-empty).

When processing q using TWINSSE$_{\text{OXT}}$, the computational costs (at both the client and the server) as well as the communication requirements between the client and the server scale linearly as $O(\gamma_0 + \gamma_1)$,

$$\gamma_0 = \sum_{k \in [n_{\text{B}}]} |q_k||\mathbf{DB}(\mathsf{mkw}^{(k)}_{i_k,j_k})|,$$

where $|q_k|$ denotes the number of meta-keywords in the conjunctive sub-meta-query q_k ($|q_k| = 0$ when q_k is empty), and

$$\gamma_1 = |\mathbf{DB}(q_1)| \cdot (\ell \in [2, L]t_\ell)\,.$$

Note that the term γ_0 is computed exactly as in the analysis of disjunctive search query overheads for TWINSSE$_{\text{OXT}}$. Moreover, the term γ_1, which represents computational and communication complexities incurred as a result of the round-2 of the CNF query processing (using OXT.GenXTag and OXT.SearchXTag), is independent of the frequencies of any of the disjunctive clauses other than the "least frequent clause" q_1.

Leakage Analysis. We state the following theorems for the leakage from TWINSSE$_{\text{OXT}}$ when processing Boolean queries in CNF form.

Theorem 7.3 (DNF Query Processing in TWINSSE$_{\text{OXT}}$)**.** *The leakage incurred by* TWINSSE$_{\text{OXT}}$ *when processing a CNF query as described above is* $\mathcal{L}^{\text{Search,CNF}}_{\text{TWINSSE}_{\text{OXT}}}$*, where for any sequence of CNF queries* $\mathcal{Q} = (q_1, \ldots, q_M)$ *such that*

$$q_m = \bigwedge_{\ell \in [L_m]} q_{m,\ell} = \bigwedge_{\ell \in [L]} (\mathsf{w}_{m,\ell,1} \vee \ldots \vee \mathsf{w}_{m,\ell,t_{m,\ell}})$$

for each $m \in [M]$*, with* $q_{m,1}$ *being (without loss of generality) the least frequent disjunctive clause for each* $m \in [M]$*, and for any pair of bucketization parameters* (n', n_{B})*, we have*

$$\mathcal{L}^{\text{Search,DNF}}_{\text{TWINSSE}_{\text{OXT}}}(\mathcal{Q}) = \left([\mathsf{RP}, \mathsf{SP}, \mathsf{EP}, \mathsf{IP}](\mathcal{Q}_{\mathsf{mkw}}), \mathcal{L}^*_{\mathsf{xtag}}\right).$$

where RP*,* SP*,* EP *and* IP *leakages for conjunctive queries are as defined in Appendix 4, and where* $\mathcal{Q}_{\mathsf{mkw},1}$ *is a sequence of (sub-)meta-queries of the form*

$$\mathcal{Q}_{\mathsf{mkw}} = \{q_{\mathsf{mkw},k,1}\}_{k \in [n_{\text{B}}]},$$

where for each $\ell \in [M]$*, we have*

$$q_{\mathsf{mkw},\ell} = \left(\bigvee_{k \in [n_{\text{B}}]} q_{\mathsf{mkw},k,1}\right) = \textsc{GenMQuery}(q_{\ell,1}, n', n_{\text{B}}),$$

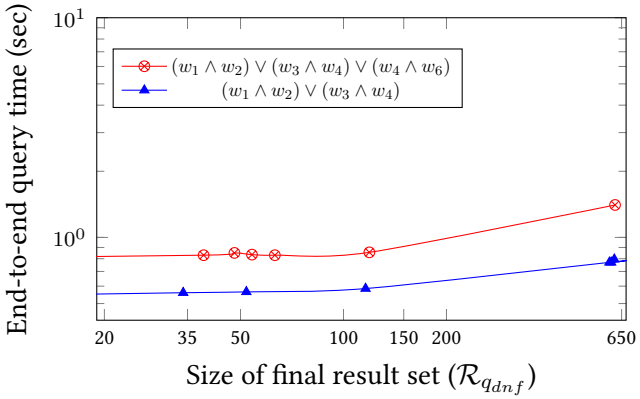

Figure 5: TWINSSE$_{\text{OXT}}$ performance with result set size on Enron dataset for DNF queries.

and, finally, we have

$$\mathcal{L}^*_{\mathsf{xtag}} = \{|\mathbf{DB}(q_{m,1}) \cap \mathbf{DB}(\mathsf{w}_{m,\ell,\ell'})|\}_{m\in[M],\ell\in[L_m],\ell'\in[t_{m,\ell}]}.$$

The proof of this theorem is again very similar to the proof of Theorem 4.1 described earlier in Appendix 5, and is hence not detailed separately.

7.3 Experimental Results over the Enron Email Dataset

We provide experimental results for CNF and DNF queries using TWINSSE$_{\text{OXT}}$ in this section. We experimented over the Enron dataset on the same platform (discussed in results of the main paper) with our implementation of TWINSSE$_{\text{OXT}}$.

DNF queries. We considered multiple queries with two clauses and three clauses with each clause having two keywords. The end-to-end query time is plotted in Figure 5, where the blue curve represents the query time for two clause queries and the red curve represents the query time for three clause queries. Observe that the query time for both two and three clause queries increase with more number of ids in the final result set. This increment can be attributed to large result size of the individual conjunctive clauses. Also note that the query time increases for three-clause queries due more conjunctive clauses and follows the same trend of increased query time with the final result size.

CNF queries. For experimenting with CNF queries, we considered two-clause queries with two keywords and three keywords per clause. Since the Enron dataset is relatively sparse in nature, with higher number of clauses in query it often results in small or empty intersection. We plotted the end-to-end query time in Figure 6 for both cases – two keyword clauses and three keyword clauses with the size of the final result set. The blue curve represents the end-to-end query time for the queries with two keywords per clause. Similarly, the red curve represents

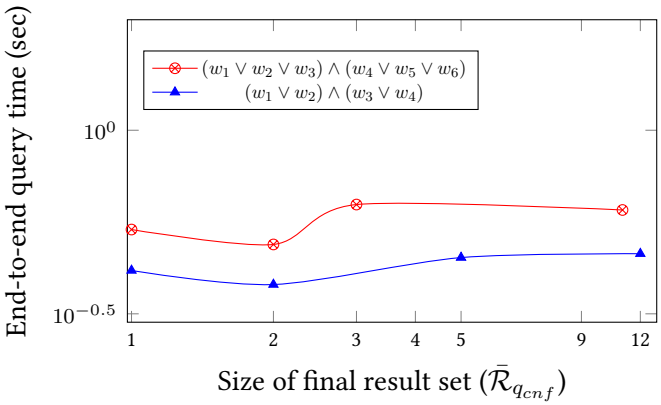

Figure 6: TWINSSE$_{\text{OXT}}$ performance with result set size on Enron dataset for CNF queries.

the end-to-end query time for queries with three-keyword clauses. Observe that, in CNF queries also, the end-to-end query time increases with the final result size, due to the increased size of the initial result set. For the three-keyword clauses, the query time is higher than the two-keyword clauses due to the increased size of the initial result set obtained by disjunctive query.

8 Experimental Results over the Wikimedia Dump

We present additional experimental results for TWINSSE$_{\text{OXT}}$ over Wikimedia databases[1] in this section. We varied the database size from $6k$ keywords ($60k$ w-id pairs in the plain index) to $80k$ keywords (8.2 million w-id pairs in the plain index), and we plot the server storage overhead in Figure 7 and performance figures in Figure 8 and Figure 9.

The comparative storage overhead plot (in log scale) in Figure 7 illustrates the quadratic storage overhead for IEX-2Lev; whereas it remains linear for TWINSSE$_{\text{OXT}}$. This storage overhead profile validates our primary contribution of our work, and also illustrates the applicability towards different databases (results on the Enron dataset is presented in the main text Section 5 of the main paper.)

9 Evaluation of Storage Overhead with Synthetic Database

We discussed in Section 5 Figure 4 (in main paper) that TWINSSE$_{\text{OXT}}$ improves significantly in terms of storage overhead than IEX-2Lev on the Enron database.

Figure 10 compares the storage overhead of TWINSSE$_{\text{OXT}}$ and IEX-2Lev on a synthetic database that follows Zipf's law and Figure 11 compares the estimated storage overhead of TWINSSE$_{\text{OXT}}$ and IEX-2Lev on a synthetic database

[1] https://dumps.wikimedia.org/enwiki/latest/

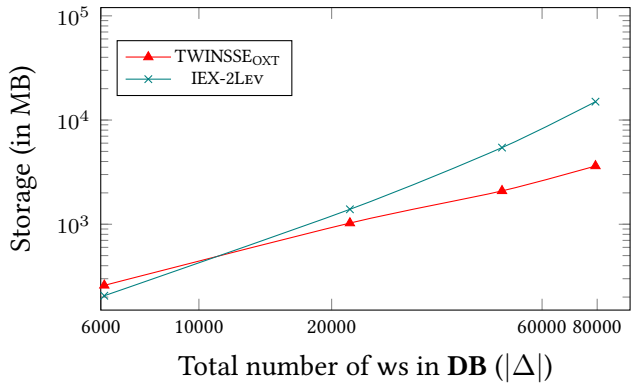

Figure 7: Server storage overhead with plain database size ($|\mathbf{DB}|$) for the Wikimedia dataset.

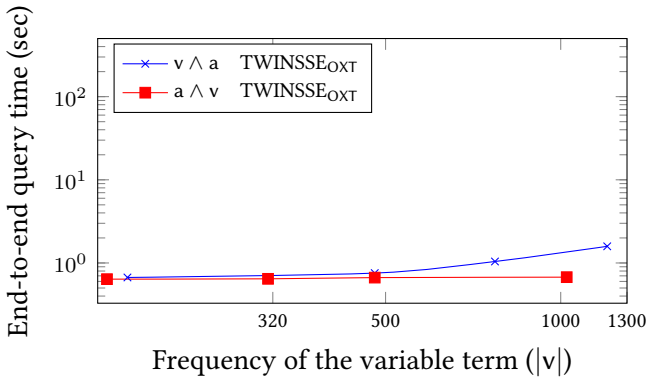

Figure 8: TWINSSE$_{OXT}$ end-to-end conjunctive query search latency vs frequency of the variable term ($|v|$) for Wikimedia dataset.

that follows a uniform distribution. These databases contain more documents per keyword than the Enron database. This implies that size of the intersections of keyword pairs is much more as compared to the Enron database. Storage overhead of IEX-2Lev hence degrades even more. Our experimental results show that IEX-2Lev incurs $70\times$ higher storage overhead than TWINSSE$_{OXT}$ for the synthetic database following Zip's law (Figure 10) and approximately $150\times$ higher storage overhead for the database following uniform distribution (Figure 11).

10 Background Material on OXT

In this section, we present a self-contained description of background material for the OXT scheme from [1].

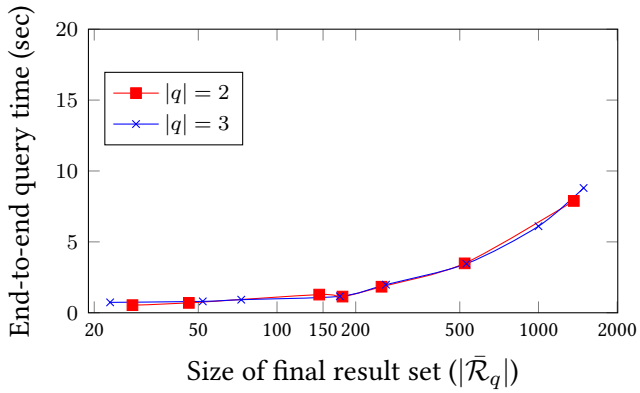

Figure 9: $\text{TWINSSE}_{\text{OXT}}$ end-to-end disjunctive query search latency vs final result size for Wikimedia dataset.

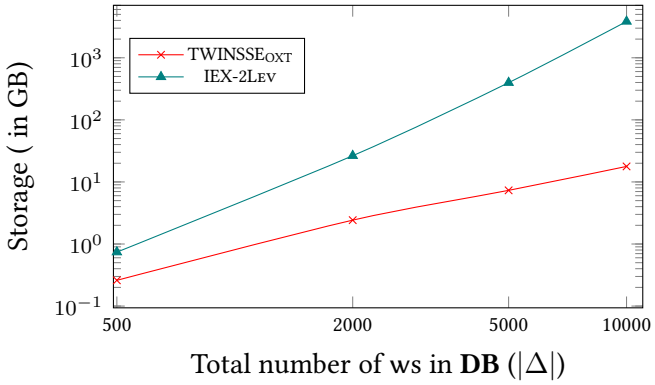

Figure 10: Server storage overhead with number of keywords in synthetic plain database ($|\Delta|$) (prepared following Zipf's distribution).

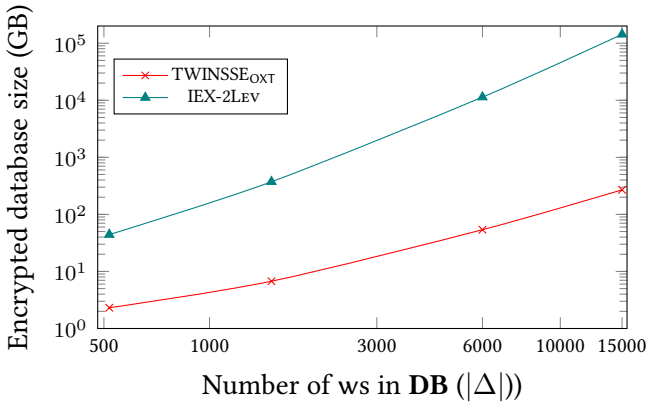

Figure 11: Server storage vs $|\Delta|$ for synthetic DB (following uniform distribution).

10.1 Cryptographic Background

This section presents the definitions and security notions for various cryptographic primitives and assumptions used in the OXT scheme.

Pseudorandom Functions. A pseudorandom function (PRF) is a polynomial-time computable function

$$F : \{0,1\}^{\lambda} \times \{0,1\}^{\ell} \longrightarrow \{0,1\}^{\ell'},$$

such that for all PPT algorithms $\mathcal{A}$, we have

$$\left|\Pr\left[\mathcal{A}^{F(K,\cdot)} = 1\right] - \Pr\left[\mathcal{A}^{G(\cdot)} = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where $K \xleftarrow{R} \{0,1\}^{\lambda}$ and G is uniformly sampled from the set of all functions that map $\{0,1\}^{\ell}$ to $\{0,1\}^{\ell'}$.

Symmetric-Key Encryption. A symmetric-key encryption scheme SKE consists of the following polynomial-time algorithms:

- $\textsc{Gen}(\lambda)$: A probabilistic algorithm that takes the security parameter λ as input and outputs a secret-key sk.
- $\textsc{Enc}(\mathsf{sk}, x)$: A probabilistic algorithm that takes as input a key sk and a plaintext x. Outputs a ciphertext c.
- $\textsc{Dec}(K, c)$: A deterministic algorithm that takes as input a key sk and a ciphertext c. Outputs the decrypted plaintext x.

A symmetric-key encryption scheme is said to be CPA-secure if for all PPT algorithms $\mathcal{A}$ and any two *arbitrary* plaintext messages x_0 and x_1, we have

$$|\Pr\left[\mathcal{A}\left(\textsc{Enc}(\mathsf{sk}, x_0)\right) = 1\right] - \Pr\left[\mathcal{A}\left(\textsc{Enc}(\mathsf{sk}, x_1)\right) = 1\right]| \leq \mathsf{negl}(\lambda),$$

where $sk \leftarrow \textsc{Gen}(\lambda)$.

Decisional Diffie-Hellman Assumption. Let $\mathbb{G}$ be a cyclic group of prime order p, and let g be any uniformly sampled generator for $\mathbb{G}$. The decisional Diffie-Hellman (DDH) assumption is that for all PPT algorithms $\mathcal{A}$, we have

$$\left|\Pr\left[\mathcal{A}\left(g, g^{\alpha}, g^{\beta}, g^{\alpha\cdot\beta}\right) = 1\right] - \Pr\left[\mathcal{A}\left(g, g^{\alpha}, g^{\beta}, g^{\gamma}\right) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where $\alpha, \beta, \gamma \xleftarrow{R} \mathbb{Z}_p^*$.

10.2 OXT Setup Algorithm

OXT EDB.SetUp in Algorithm 3 takes as the security parameter λ and the plaintext database **DB** as input, and generates the encrypted database **EDB** and master key mk associated with that database. The encrypted database **EDB** is stored on the remote server, and the client uses the master key to search over the encrypted database. OXT uses a data structure called TSet to store the encrypted data. Detailed construction of TSet is available in OXT paper [1].

Algorithm 3 Setup algorithm of OXT

Input: 1^λ, **DB**
Output: $\mathsf{mk}, param$, **EDB**

function EDB.SETUP(1^λ, **DB**)
  Initialise $T \leftarrow \phi$ indexed by keywords Δ
  Select key K_S for PRF F
  Select keys K_I, K_Z, K_Z for PRF F_p
  Initialise **EDB** $\leftarrow \{\}$
  Initialise $\mathsf{XSet} \leftarrow \{\}$
  for $w \in \Delta$ do **do**
    Initialise $t \leftarrow \{\}$
    Compute $k_e \leftarrow F(K_S, \mathsf{w})$
    Set counter $c \leftarrow 1$
    for $\mathsf{id} \in$ **DB**(w) do **do**
      Compute $xid \leftarrow F_p(K_I, \mathsf{id})$
      Compute $z_w \leftarrow F_p(K_Z, \mathsf{w}||c)$
      Compute $y_{\mathsf{id}} \leftarrow xid \cdot z_w^{-1}$
      Compute $e_c \leftarrow Sym.Enc(k_e, \mathsf{id})$
      Set $xtag \leftarrow g^{F_p(K_X, \mathsf{w}) \cdot xid}$
      Add $xtag$ to XSet
      Append (y_{id}, e_c) to t and set $c \leftarrow c + 1$
    Set $T[\mathsf{w}] \leftarrow t$
  Compute $\{\mathsf{TSet}, K_T\} \leftarrow \mathsf{TSet.SetUp}(T)$ ▷ *(See [1] for TSet routines)*
  return $\mathsf{mk} = \{msk, K_S, K_I, K_Z, K_X, K_T\}$, **EDB** $= (\mathsf{TSet}, \mathsf{XSet})$

10.3 OXT Search Algorithm

EDB.Search of OXT is a two-party algorithm (Algorithm 4) involving client and hosting server. OXT Search takes the query keywords $q = \{\mathsf{w}_1, \cdots, \mathsf{w}_n\}$, master key mk, and **EDB** as input and outputs the result of the query R_q.

Algorithm 4 Search algorithm of OXT

Input: $\mathsf{mk}, param, q = (\mathsf{w}_1 \wedge \ldots \wedge \mathsf{w}_n), \mathbf{EDB}$
Output: Result R_q

function EDB.SEARCH($\mathsf{mk}, param, q, \mathbf{EDB}$)
- Client's inputs are $(\mathsf{mk}, param, q)$ and server's inputs are $(param, \mathbf{EDB})$
- Client initialises $R_q \leftarrow \{\}$ and computes $stag \leftarrow \mathsf{TSet.GetTag}(K_T, \mathsf{w}_1)$
- Client sends $stag$ to the server
- Server recovers $\mathbf{EDB}(1) = \mathsf{TSet}$. starts accepting $xtokens$ computed by client as follows:
- **for** $c = 1$: until server sends $stop$ **do**
  - Client computes $f_{\mathsf{w}_1} \leftarrow F_p(k_Z, \mathsf{w}_1 || c)$
  - **for** $l = 2 : n$ do **do**
    - Client computes $xtoken[c, l] \leftarrow g^{f_{\mathsf{w}_1} \cdot F_p(k_X, \mathsf{w}_l)}$
  - Client sets $xtoken[c] \leftarrow (xtoken[c, 2], \ldots, xtoken[c, n])$.
  - Client send $xtoken[c]$ to server.
- Server initialises $\mathcal{E} \leftarrow \{\}$.
- Server computes $t \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, stag)$,▷ *(See [1] for TSet routines)*
- **for** $c = 1 : |t|$ do **do**
  - Server recovers (y_{id}, e_c) from c-th component of t.
  - **for** $l = 2 : n$ do **do**
    - Server computes $xtag = xtoken[c, l]^{y_{\mathsf{id}}}$
  - If $\forall l \in [2, n]$, $xtag \in \mathsf{XSet}$, then send e_c to the client.
  - When last tuple in t is reached, send $stop$ to client and halt.
- Client computes $K_e \leftarrow F(K_S, \mathsf{w}_1)$.
- Client computes $\mathsf{id}_c \leftarrow Sym.Dec(K_e, e_c)$, and adds id_c to R_q for all e_c received.
- **return** R_q