MARS: Low-Leakage Multi Adversarial Owner and Reader Replication-free Searchable Encryption from Private Information Retrieval

Benjamin Fuller*, Arinjita Paul†, Maryam Rezapour‡, Ronak Sahu§, Amey Shukla¶

October 28, 2025

Abstract

In searchable encryption, a data owner outsources data to a server while allowing efficient search by clients. A multimap associates keywords with a variable number of documents. We consider the setting with multiple owners and multiple clients (Wang and Papadopolous, Cloud Computing 2023). The goal is for each owner to store a multimap and grant access to clients. Prior work shares three weaknesses:

1. Restricting patterns of adversarial behavior,
2. Duplicating any data shared with a new client, and
3. Leaking each client's access pattern and share pattern.

We present MARS, the first SSE for multiple owners and clients that considers security for an arbitrary collection of adversarial parties. Our scheme only leaks the volume of the result size and the number of requested keywords, both of which can be padded. No data is replicated.

Our scheme combines 1) private information retrieval (PIR) to protect search patterns, 2) efficient delegation of the ability to index keywords and decrypt records, and 3) tabulation hashing to allow a single query for locations associated with a keyword. The first two items can be thought of as a keyword unkeyed PIR where the data owner gives the client the identifiers for individual keywords and keys to decrypt records.

Our system is implemented on multimaps up to size 24 million (the Enron dataset) with total time of 1.2s for keywords that match 100 documents. This is in comparison to a time .500s for Wang and Papadapolous, which replicates data and has access, sharing, and query equality leakage.

Storage overhead is a factor of 6.6. Our implementation uses FrodoPIR as the underlying PIR. Our system can incorporate batch or doubly-efficient unkeyed PIR as their performance improves.

1 Introduction

Searchable encryption [SWP00, BHJP14, FVY+17, KKM+22, AGY+22, LMM+23] considers three parties: a data owner, $\mathtt{Ow}$, a server, $\mathtt{S}$, and a client, $\mathtt{C}$. We consider search over databases organized as a multimap [KM18, KMO18, AG22, APP+23, GPPW24], denoted as $\mathtt{MM}$, where keywords are associated with a variable number of documents. For each keyword w, one associates $\mathsf{Docs}_{\mathsf{w}}$ into the multimap, where $\mathsf{Docs}_{\mathsf{w}}$ is a variable length list of the documents with keyword w. That is, $\mathtt{MM}[\mathsf{w}] = \mathsf{Docs}_{\mathsf{w}}$.

At search time, one looks up some w from the multimap. The goal is for a data owner, denoted $\mathtt{Ow}$, to outsource multimap $\mathtt{MM}$ to the server, denoted $\mathtt{S}$, so that multiple clients, denoted $\mathtt{C}$, can efficiently access the data with minimal involvement from $\mathtt{Ow}$. We consider the scenario where there are multiple clients and owners known as the multi-reader and multi-writer setting respectively [BHJP14].

*University of Connecticut. Email: benjamin.fuller@uconn.edu
†Niobium Microsystems/University of Connecticut. Email: arinjita.paul@gmail.com
‡University of Connecticut. Email: maryam.rezapour@uconn.edu.
§University of Connecticut. Email: ronak.sahu@uconn.edu.
¶University of Connecticut. Email: amey.shukla@uconn.edu.

Prior Work Popa et al. [PZ13] define a primitive they call multi-key searchable encryption. There are three algorithms:

1. Setup: An owner, Ow encrypts MM and sends it to S.
2. Share: Ow delegates to a C the ability to access data. This procedure often involves the S.
3. Query: A client C searches for w over the data that has been shared with them.

The first generation of multi-key schemes [PZ13, PSV+14, KOR+16] demonstrate joint access pattern leakage: identifiers for a set of documents that match each client's query are visible to the server. These identifiers are persistent across queries and clients. By colluding with an adversarial C or Ow that can issue their own queries, this leakage can be devastating [GMN+16, VRMÖ17, HSWW18]. The second generation of solutions [HSWW18, WP21, WP23] addresses this issue by duplicating data when it is shared to each C decoupling different clients' access patterns. This approach causes storage to grow, server storage for a single owner may reach $O(|\texttt{MM}|m)$ for m clients.[1]

Eliminating Leakage in the Single Client Setting Attacks on access pattern leakage have proved devastating in the single-reader/writer setting [IKK12, CGPR15, KKNO16, WLD+17, GSB+17, GLMP18, MT19, KE19, KPT20, GPP23]. New schemes have zero or very low-leakage through the judicious use of oblivious techniques [GKL+20, MVA+23, DPP+16, GPP23, PPYY19, GPPW24, GKM21, KMO18, KMPP22, DPPS20, LXY+23] such as oblivious RAM (ORAM) [Gol87, GO96] and private information retrieval (PIR) [KO97, CG97, CKGS98, BIM00a, CHR17, BIPW17, LMW23, CD24]. Such oblivious techniques reveal only volume information, which can be padded [GKL+20, MVA+23].[2]

A standard design of a secure MM is to build two maps, called VolMap, for volume map, and ValMap, for value map, where

$$\texttt{VolMap}[\mathsf{w}] = |\texttt{MM}[\mathsf{w}]|,$$
$$\texttt{ValMap}[(\mathsf{w}, i)] = \texttt{MM}[\mathsf{w}][i], 1 \leq i \leq |\texttt{MM}[\mathsf{w}]|.$$

We formalize this standard design in Section 3. These two maps can be secured with different techniques, and their respective performance and leakage "adds." MUSSE [WP21, WP23] present three strategies for securing a Map:

1. Local: Local storage by C,
2. MUSSE: Using symmetric techniques with access pattern leakage. (There is additional leakage related to timing of sharing, see Section 3.)
3. OMap: A tree of Map keywords in ORAM [WNL+14].

Unfortunately, all three solutions replicate the data. Their most secure construction uses an OMap for VolMap and MUSSE for ValMap.

Allowing unlimited access using keyword PIR As an alternative, one could use keyword unkeyed PIR [CGN97, MK22, PSY23, CD24] to prepare an array that can be accessed by any user. One can easily encrypt an array before applying PIR to hide all information about the array other than length. The owner Ow can allow C to decrypt parts of this array, see Section 4. Keyword unkeyed PIRs [MK22, CD24] explicitly or implicitly create a Translate function between keywords and indices. The Translate function has to be public. As an example, Mahdavi and Kerschbaum [MK22] make the set of keywords in the map public. To use such tools in searchable encryption, one needs to create a level of indirection between keywords and the values stored in Translate. To make this distinction clear we use three terms: keywords which are the values in the map (w), identifiers (Id) which are the keywords indexed by Translate, and indices (i) which are the locations to be retrieved from the PIR.

[1] Hamlin et al. [HSWW18] present a second construction where the size of the sharing is independent of the number of shared values that relies on public-coin different inputs obfuscation [IPS15]. We restrict our attention to standard cryptographic tools that can be implemented on benchmark datasets.

[2] We don't discuss volume padding further; previous work on volume padding is applicable. For example, the volume of an item can be extended to the next power of 2.

1.1 Our Contribution

We present three contributions in this work.

1. A keyword unkeyed PIR that combines a PIR with a data independent packing (DIP) [BBF+21] for translating identifiers into indices. The DIP guarantees that the output size of the array stored in the PIR is independent of the Map being stored and that the packing succeeds with overwhelming probability. We call this approach for protecting a map KPIR. We note that the guarantees of the DIP suffice if the Map is specified before the DIP is selected, stronger primitives are needed if the map depends on the DIP [Yeo23].

   Ow can efficiently send to clients $(\mathsf{w}, \mathsf{Id})$ and $\mathsf{w}, K_{\mathsf{w}}$ pairs to allow the C to find and decrypt the values corresponding to a w respectively. Such a solution only leaks two things:

   (a) To the server, the size of the multimap, the number of distinct keywords in the MM, and the volume, $|\mathtt{MM}[\mathsf{w}]|$, for each searched w. In our implementation, ρ items are packed together in a single PIR index so there is some padding, these values can be additionally padded.

   (b) To an owner, information dependent on what policy it enforces for delegating key to clients. We present two approaches where keys are sent after the owner learns the keyword and where this interaction occurs through an oblivious pseudorandom function (OPRF) [CHL22]. We call such mechanisms *access control* mechanisms.

2. An implementation of KPIR to protect VolMap and ValMap for a two round search where the only leaked information is the padded volume of the search term. Identifiers and keys are sent from owner to client using a OPRF. Using FrodoPIR [DPC23] as the underlying PIR, we implement the above on the Enron dataset [KY04]. The Enron dataset has $2^{24.5}$ keyword, document pairs and 2^{18} distinct keywords. Unfortunately, a natural implementation of the above ideas is too slow.

   The solution requires two networks rounds and roughly 20s of computation time. For a keyword with 100 matching documents, on a 100Mb network, the network latency is roughly 12 seconds for an overall time of 32s. This is clearly not competitive with the most secure version of MUSSE which requires roughly .5 seconds. In all of our comparisons with MUSSE, for MUSSE we only count delay due to roundtrips, assuming that latency due to bandwidth and computation time are 0.

3. Novel cryptographic techniques when the PIR is *shift friendly*, meaning the server can oblivious change the searched position. **For such PIRs, we show how to combine tabulation hashing, shift friendly PIR, and sharding to reduce overall search time from 32s to 1.2s.** Before introducing tabulation hashing, we introduce two technical details about KPIR. For most single server PIR approaches, the query is much larger than the response which is able to collapse from a large array to a few elements of an array. The size of the query causes most of the 12s network delay. Our DIP is based on Cuckoo hashing. For an $(\mathsf{w}, i) \in \mathtt{VolMap}$ denoting the ith matching document for w the Translate computes two hashes H_1, H_2 on $\mathsf{Id}_{\mathsf{w},i}$ to find the relevant positions.

   (a) We replace the hashes in the DIP with tabulation hashes [PT12] where instead of $H(w, i)$, the relevant positions are $H(\mathsf{w}) + H'(i)$ for $1 \leq i \leq \mathtt{Vol}$ for hash functions H, H'.

   (b) We show for certain PIRs, including FrodoPIR [DPC23] and DEPIR [LMW23], from the value $\mathsf{Query}(x)$ the server can shift the position by a public value y to yield $\mathsf{Query}(x + y)$. This allows the server to obliviously shift $\mathsf{Query}(H(w))$ into $\mathsf{Query}(H(\mathsf{w}) + H'(i))$. This allows the server to simultaneously craft a response for all (w, i) from just $\mathsf{Query}(H(\mathsf{w}))$. There are negative results on the collision probability of tabulation hashing in comparison to random hashing [Tho17]. However, in our testing, this change does not harm the quality of packing or the number of documents that cannot be packed. (Many packing based techniques have a small stash, in our experiments the stash size never exceeds 1000.)

   (c) Sharding solves the computation problem; tabulation hashing solves the bandwidth problem.

      i. The techniques in isolation don't bring much improvement. Using just sharding, the query time is 8s and using just tabulation hashing, time is 26s.

      ii. Together the techniques reduce query time from 32s to 1.2s.

A. For 32 shards sharding reduces the computation time from 20 to .6s,
B. At one shard tabulation hashing reduces BW from 93MB to 19. At 32 shards it reduces BW from 94MB to 1.6MB causing an overall network delay of .6s.

(d) We additionally show for FrodoPIR, owners can use reuse some lattice objects and Translate to allow clients to issue a single query that works for all $\mathtt{Ow}$.

In our final system, search takes 1.2 seconds, only twice that of MUSSE which has extensive leakage of 1) access pattern [CGPR15], 2) query equality pattern, 2) sharing pattern which is leakage related to when a keyword was shared with a client.

As a secondary approach, if one desires a single round solution, in Appendix A we show how to use MUSSE in a mode that is response-revealing and considers adversarial owners to protect $\mathtt{VolMap}$ which using $\mathtt{KPIR}$ to protect $\mathtt{ValMap}$. Of course, such a combination does leak information on $\mathtt{ValMap}$ to the server and through adversarial clients.

1.1.1 Further Design Details

DIP A DIP translates the $\mathtt{Map}$ into a sequence of locations and a stash with the guarantee that with overwhelming probability each item can "fit." The DIP produces a key, K_{DIP}, used to map keywords w's into a set of locations in the array. The DIP can be viewed as a Cuckoo hash where K_{DIP} is two independently sampled hashes where the packing succeeds with overwhelming probability.[3] Dummy ciphertexts are added in locations that are not used. The entire stash is accessed with every query (it can be saved by a client after the first query). In our setting K_{DIP} is publicly released and parts of the encrypted array can be decrypted by adversarial clients. However, the w to Id mapping is through a (O)PRF that is managed by the data owner, meaning a public K_{DIP} does not impact security.

PIR Our work builds on unkeyed PIR.[4] The DIP creates a Translate function mapping keywords to indices. Our implementation uses FrodoPIR [DPC23], a linear scan PIR based on lattices. For the Enron Corpus, when a keyword is associated with 100 documents, our query processing time is 0.600 seconds with 32 shards, requiring 94MB of bandwidth while tabulation hashing reduces this bandwidth to 1.6MB. Unfortunately, current batch PIR [BIM00b, IKOS04, ACLS18] and doubly efficient unkeyed [OPPW24] are slower than FrodoPIR except for keywords with many associated documents (thousands see Section 5).

Summary and Future Work Our security is qualitatively stronger than prior work, the only leakage of our system is the (padded) volume of search term and the number of shared w's with each client. See Table 2 for a comparison of security. Many previous techniques for hiding volume apply in our setting as well [DPPS20, GKL+20, MVA+23].

Our use of PIR does mean higher Search overhead than prior work, see Table 1. Our optimizations bring performance inline with prior work. We present a proof of concept implementation (Section 5) with implementation on random data and the Enron Corpus with sizes of multimaps up to $|\mathtt{MM}| = 2^{24.5}$. Storage overhead is roughly 6.6 over the unprotected $\mathtt{MM}$. We encourage development of a variety of access control mechanisms. We briefly discuss two further candidates:

1. BlindSeer [FVK+15] evaluates a policy, query pair using garbled circuits between the owner and client.
2. One can use Labeled Private Set Intersection (LPSI) [CHLR18, CLR17, KKRT16, PSWW18, UCK+21] where $\mathtt{Ow}$ prepares a set of $(\mathsf{w}, K_{\mathsf{w}})$'s where K_{w} is a decryption key for documents associated with w, and $\mathtt{C}$ holds w_q. Then for any w where $\mathsf{w} = \mathsf{w}_q$, $\mathtt{C}$ receives K_{w} which allows them to decrypt the documents corresponding to w. LPSI is equivalent to batch keyword PIR [CHLR18], so this would correspond to a PIR interaction between the client and owner, albeit on an array whose size is proportional the number of keywords the owner is willing to grant to the client, which is of size at most $|\mathtt{VolMap}|$. This approach also works for sending the identifiers to client.

[3] In our and previous definitions of multi-key searchable encryption, the multimaps are specified before hash functions are sampled so the two objects are independent. In an adaptive definition where the $\mathtt{MM}$ depends on the hash functions, one needs a stronger primitive, see discussion in Yeo [Yeo23].

[4] The terminology unkeyed is only used in DE-PIR where there is a preprocessing step to allow for sublinear scans.

Scheme	Storage		Computation		Communication		Rounds
	Client	Server	Search	AcCtrl	Search	AcCtrl	
[PPY18]	$\lvert\mathsf{Docs}\rvert$	$\lvert\mathtt{MM}^{\mathtt{Ow}}\rvert$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	1
[HSWW18]	1	$\sum_{\mathsf{C}} \lvert\mathtt{MM}^{\mathsf{C}}\rvert$	$\lvert\mathsf{Docs}^{\mathsf{C}}\rvert$	1	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	1
[WP21] - NFNU	1	$\sum_{\mathsf{C}} \lvert\mathtt{MM}^{\mathsf{C}}\rvert$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$
[WP21] - FU	κ	$\sum_{\mathsf{C}} \lvert\mathtt{MM}^{\mathsf{C}}\rvert$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$	1	1
[WP21] - FNU	1	$\sum_{\mathsf{C}} \lvert\mathtt{MM}^{\mathsf{C}}\rvert$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert + \log^2 \kappa$	$\lvert \log^2 \kappa \rvert$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert + \log^2 \kappa$	$\log^2 \kappa$	$\log \kappa$
MARS w/ DEPIR	1	$(\lvert\mathtt{MM}\rvert)^{1+o(1)}$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert\lvert\mathtt{MM}\rvert^{o(1)}$	1	$\lvert\mathtt{MM}\rvert^{o(1)}$	1	2
MARS w/ FrodoPIR	1	$\lvert\mathtt{MM}\rvert$	$\lvert\mathsf{Docs}_{\mathsf{w}}\rvert\lvert\mathtt{MM}\rvert$	1	$\lvert\mathtt{MM}\rvert$	1	2

Table 1: Efficiency comparison with prior works. $\mathtt{MM}^{\mathsf{C}}$ is the size of the database shared with C. $\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$ is the number of documents that are associated with keyword w. $\kappa := \lvert\mathtt{VolMap}\rvert$ represents the total number of distinct keywords. All notation is O and dependent on security parameter λ but this is omitted, except for roundtrips, if this is a constant the constant is listed. We defer comparison of security to Table 2. We slightly MUSSE to share keywords instead of documents, so our asymptotics do not exactly match theirs.

ValMap	$\mathtt{Ow}_{\mathcal{A}}$	Leakage	$\text{Owner}_{\mathcal{A}}$ Leakage	VolMap	Replication	Additional Leakage
MUSSE [WP21]	✗	ValMap Access Pattern, Share pattern	Share requests	Local OMap MUSSE KPIR	ValMap & VolMap ValMap	VolMap Access Pattern, Most Recent Share
KPIR	✓	Query Volume	Policy Depd.	KPIR	None	

Table 2: Comparison of candidate instantiations. $\mathtt{Ow}_{\mathcal{A}}$ indicates if the scheme considers adversarial owners. $\text{Client}_{\mathcal{A}}$ & $\text{Owner}_{\mathcal{A}}$ leakages are only on interactions involving these adversarial parties. All schemes leak $\lvert\mathtt{MM}\rvert = \lvert\mathtt{ValMap}\rvert$, $\kappa := \lvert\mathtt{VolMap}\rvert$, and for a query w, the size of $\lvert\mathsf{Docs}_{\mathsf{w}}\rvert$.

Organization The rest of this work is organized as follows, Section 2 covers preliminaries including several new definitions, Section 3 presents our system, Section 4 presents our access control mechanisms, and Section 5 presents our implementation and experimental results. Code is available at our GitHub repository [FPR+25] for both tabulation and random hashing.

2 Preliminaries

We consider protocols involving up to three parties: a data owner $\mathtt{Ow}$, server $\mathtt{S}$, and a client $\mathtt{C}$. At most two parties are involved in each protocol and they are listed in the above order. As an example of notation, for an interactive protocol Prot between two parties A and B, we use notation

$$\begin{pmatrix} o_{\mathsf{A}} \\ o_{\mathsf{B}} \end{pmatrix} \leftarrow \mathsf{Prot}^{\mathsf{A},\mathsf{B}} \begin{pmatrix} i_{\mathsf{A}} \\ i_{\mathsf{B}} \end{pmatrix}$$

with $i_{\mathsf{A}}, o_{\mathsf{A}}, i_{\mathsf{B}}, o_{\mathsf{B}}$ denoting A and B's inputs and outputs respectively. Throughout, we consider m clients and n owners but only a single client or owner participates in any protocol. The numbers n, m are not input to any protocol initialization. If a party does not participate in a protocol, they are elided. For variables, parties are listed in superscript while other indexing (such as sequence) is done in subscript.

A map $\mathtt{Map}$ is an association between keywords w and a document $\mathsf{Doc}_{\mathsf{w}}$. A multimap $\mathtt{MM}$ associates keywords w and $\mathsf{Docs}_{\mathsf{w}}$, the set of documents associated with each keyword. We assume that each keyword is over a universe $\mathcal{W}$ and that each document is of the same length η. Of course, documents can be split and padded into chunks of size η. The total size of $\mathtt{MM}$ is $\ell \cdot \eta := \sum_{\mathsf{w} \in \mathcal{W}} \lvert\mathsf{Docs}_{\mathsf{w}}\rvert$ where ℓ is the number of $(\mathsf{w}, \mathsf{Doc})$ pairs. In the above syntax, documents are counted once for each of their associated keywords. Looking ahead we assume each Doc is an integer identifier, see Section 5 for discussion on storing the entire document inside $\mathtt{MM}$. This is common in searchable encryption, however one needs to consider how to look up the documents from an array in a secure way [GPPW24].

For a multimap and some arbitrary binary operator $\leq$, $\mathsf{Arr}_{\mathtt{MM}}$ outputs the pairs $(\mathsf{w}, \mathsf{Doc})$ and is indexed starting from position 1 where $\mathsf{Doc} \in \mathsf{Docs}_{\mathsf{w}}$, and $\mathsf{w} \in \mathtt{MM}$. The pairs $(\mathsf{w}_1, \mathsf{Doc}_1), (\mathsf{w}_2, \mathsf{Doc}_2)$ are ordered by keyword using $\leq$. In the case of pairs $(\mathsf{w}, \mathsf{Doc}_1), (\mathsf{w}, \mathsf{Doc}_2)$, they are ordered by comparing $\mathsf{Doc}_1, \mathsf{Doc}_2$ using $\leq$. We assume that the list $\mathtt{MM}[\mathsf{w}]$ is returned in the same order as in $\mathsf{Arr}_{\mathtt{MM}}$ (using the comparator $\leq$). For a

Notation	Meaning
$\mathtt{Map}$	Map
$\mathtt{MM}$	Multimap
Arr	Array representation of $\mathtt{Map}$ or $\mathtt{MM}$
η	Length of each document
ℓ	Number of keyword-document pairs
κ	Number of keywords in $\mathtt{MM}$
w	Keyword
$\mathsf{Docs}_{\mathsf{w}}$	All documents associated with keyword w
$\mathtt{Shr}$	Keywords that are requested to be shared
$\mathtt{Plcy}$	Policy
q	Number of queries/policy changes, superscripted by parties
m	Number of clients
n	Number of owners
KOwn	Owner's key
K	Key, superscripted by parties, subscripted by map, vector, or the scheme being encrypted.
ρ	Size of each array element stored in PIR
q	PIR query
ans	Server's response to PIR query
μ	Number of buckets required by the DIP
s	Number of possible positions for each w in DIP
Pos	Bucket indices from DIP
σ	Size of the Stash in DIP

Table 3: Summary of Notation

vector $\vec{a}$ of length ℓ and permutation $\pi : [1, \ell] \to [1, \ell]$ we use $\pi(\vec{a})$ to be a new array in the order specified by π. We use similar notation for a $\mathtt{Map}$.

All Arr elements consist of two parts, the first part is a w and the second part is a Doc or a ciphertext, c. We use notation $\mathsf{Arr}[i][\mathsf{w}]$ and $\mathsf{Arr}[i][\mathsf{Doc}]$ or $\mathsf{Arr}[i][c]$ respectively for the two parts.

Simulators are assumed to be stateful, keeping inputs from their previous invocations.

Notation is summarized in Table 3.

2.1 Searchable Encryption Definitions

We first define a multi-key searchable encryption where sharing is done before search.

Definition 1. *(Multi-Key Searchable Encryption) A tuple* $(\mathsf{KeyGen}^{\mathtt{Ow}}, \mathsf{Setup}, \mathsf{Search}, \mathsf{Share})$ *of PPT algorithms is a Multi-Key Searchable Encryption Scheme (*MKSE*) if the following holds:*

- $\mathsf{KOwn}^{\mathtt{Ow}} \leftarrow \mathsf{KeyGen}^{\mathtt{Ow}}(1^\lambda)$*: Takes as input the security parameter* 1^λ *and returns a data key* $\mathsf{KOwn}^{\mathtt{Ow}}$*.*
- $\begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}} \\ \mathtt{EMM} \end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}} \begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{MM} \\ \perp \end{pmatrix}$*: Create an encrypted structure stored on the server.* KOwn *is updated to include the necessary state.*
- $\begin{pmatrix} \perp \\ K^{\mathtt{C}} \end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}} \begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{Plcy} \\ K^{\mathtt{C}}, \mathtt{Shr} \end{pmatrix}$*:* $\mathtt{C}$ *requests a list of keywords* $\mathtt{Shr}$*. We assume that* $\mathtt{Plcy}$ *is a function of* $\mathtt{Shr}$ *and has any information needed about the encrypted multimap to make policy decisions. The value of* $K^{\mathtt{C}}$ *is updated. On first call,* $K^{\mathtt{C}} = \perp$*.*
- $\begin{pmatrix} \perp \\ \mathsf{Res} \end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}} \begin{pmatrix} \mathtt{EMM} \\ K^{\mathtt{C}}, \mathsf{w} \end{pmatrix}$*:* $\mathtt{C}$ *searches for keyword* w *and receives a response,* Res*.*

Correctness *Let λ be a parameter. Consider the correctness experiment shown in Figure 1. The* $\mathsf{MKSE} := \mathsf{MKSE}(\lambda)$ *is correct if for all PPT adversaries $\mathcal{A}$ and all $q := q(\lambda) = \mathtt{poly}(\lambda)$, it is true that*

$$\Pr[\mathsf{Exp}_{\mathtt{Correct},\mathcal{A},\mathsf{MKSE}}(1^\lambda, q)] \geq 1 - \mathtt{ngl}(\lambda).$$

Security *Let λ be a security parameter. Consider the security experiment shown in Figure 2. The* $\mathsf{MKSE} := \mathsf{MKSE}(\lambda)$ *is secure if for all PPT adversaries $\mathcal{A}$ there exists a PPT* Sim *such that for all $q := q(\lambda) = \mathtt{poly}(\lambda)$ it is true that*

$$\left| \begin{matrix} \Pr[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{MKSE}}(1^\lambda, q) = 1] \\ -\Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{Sim},\mathsf{MKSE}}(1^\lambda, q) = 1] \end{matrix} \right| \leq \mathtt{ngl}(\lambda).$$

$\underline{\mathsf{Exp}_{\mathtt{Correct},\mathcal{A},\mathsf{MKSE}}(1^\lambda, q)}$:

$\mathcal{A}$ sends to $\mathcal{C}$:

1. A set of client identities $\mathtt{Clients} = \{\mathtt{C}^j\}_{j=1}^m$ and data owners identities $\mathtt{Owners} = \{\mathtt{Ow}^i\}_{i=1}^n$.
2. For each $(\mathtt{Ow}, \mathtt{C})$, $q^{\mathtt{Ow},\mathtt{C}}$ where $q^{\mathtt{Ow},\mathtt{C}} < q$.
3. For each $(\mathtt{Ow}, \mathtt{C})$ and step $1 \leq i \leq q^{\mathtt{Ow},\mathtt{C}}$, queries $\mathsf{w}_i^{\mathtt{Ow},\mathtt{C}}$, policies $\mathtt{Plcy}_i^{\mathtt{Ow},\mathtt{C}}$, and share requests $\mathtt{Shr}_i^{\mathtt{Ow},\mathtt{C}}$.
4. For each $\mathtt{Ow} \in \mathtt{Owners}$, $\mathtt{MM}^{\mathtt{Ow}}$.

$\mathcal{C}$ performs:

1. For each $\mathtt{Ow}$, generates

$$\mathsf{KOwn}^{\mathtt{Ow}} \leftarrow \mathsf{KeyGen}_{\mathtt{Ow}}(1^\lambda),$$
$$\begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}} \\ \mathtt{EMM}^{\mathtt{Ow}} \end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}} \begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{MM}^{\mathtt{Ow}} \\ \perp \end{pmatrix}.$$

2. For each pair $(\mathtt{Ow}, \mathtt{C})$, $1 \leq i \leq q^{\mathtt{Ow},\mathtt{C}}$, run

$$\begin{pmatrix} \perp \\ \mathsf{K}^c \end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}} \begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{Plcy}_i^{\mathtt{Ow},\mathtt{C}} \\ K^{\mathtt{C}}, \mathtt{Shr}_i^{\mathtt{Ow},\mathtt{C}} \end{pmatrix}$$
$$\begin{pmatrix} \perp \\ \mathsf{Res}_i^{\mathtt{Ow},\mathtt{C}} \end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}} \begin{pmatrix} \mathtt{EMM}^{\mathtt{Ow}} \\ K^{\mathtt{C}}, \mathsf{w}_i^{\mathtt{Ow},\mathtt{C}} \end{pmatrix}$$

   where $K^{\mathtt{C}}$ is initialized to $\perp$.

Output 1 if for all $(\mathtt{Ow}, \mathtt{C})$, $1 \leq i \leq q^{\mathtt{Ow},\mathtt{C}}$,

$$\mathsf{Res}_i^{\mathtt{Ow},\mathtt{C}} = \begin{cases} \mathsf{Docs}^{\mathtt{Ow},\mathtt{C}}[\mathsf{w}_i^{\mathtt{Ow},\mathtt{C}}] & \exists j \leq i, \begin{matrix} \mathsf{w}_i^{\mathtt{Ow},\mathtt{C}} \in \mathtt{Shr}_j^{\mathtt{Ow},\mathtt{C}} \\ \wedge \mathtt{Plcy}_j^{\mathtt{Ow},\mathtt{C}}(\mathtt{Shr}_j^{\mathtt{Ow},\mathtt{C}}) = 1 \end{matrix}, \\ \perp & \text{otherwise.} \end{cases}$$

Figure 1: Correctness Experiment for MKSE. We let q represent the maximum number of queries and share requests both of which can be empty for some slot $i, 1 \leq i \leq q$.

Discussion The definition differs from previous definitions. Unlike Hamlin et al. [HSWW18], we use simulation security. Unlike Wang and Papadopoulos [WP21] and Chamani et al. [CWP+21], we consider

$\underline{\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{MKSE}}(1^\lambda, q)}$:
$\mathcal{A}$ sends to $\mathcal{C}$:

1. A set of clients $\mathtt{Clients} = \{\mathtt{C}^j\}_{j=1}^n$ and owners $\mathtt{Owners} = \{\mathtt{Ow}^i\}_{i=1}^m$,
2. A subset of corrupted parties $\mathtt{Owners}_\mathcal{A} \subseteq \mathtt{Owners}$ and $\mathtt{Clients}_\mathcal{A} \subseteq \mathtt{Clients}$ along with keys KOwn for each $\mathtt{Ow} \in \mathtt{Owners}_\mathcal{A}$.
3. For each $(\mathtt{Ow}, \mathtt{C})$, $q^{\mathtt{Ow},\mathtt{C}}$ where $q^{\mathtt{Ow},\mathtt{C}} < q$.
4. For each $\mathtt{Ow} \in \mathtt{Owners}$, $\mathtt{MM}^{\mathtt{Ow}}$.

$\mathcal{C}$ performs:

1. For $\mathtt{Ow} \notin \mathtt{Owners}_\mathcal{A}$, $\mathsf{KOwn}^{\mathtt{Ow}} \leftarrow \mathsf{KeyGen}^{\mathtt{Ow}}(1^\lambda)$,
2. For each $\mathtt{Ow}$, run
$$\begin{pmatrix} \mathsf{KOwn} \\ \mathtt{EMM}^{\mathtt{Ow}} \end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}} \begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{MM}^{\mathtt{Ow}} \\ \perp \end{pmatrix}.$$
Send the view of parties in $\mathtt{Owners}_\mathcal{A} \cup \mathtt{Clients}_\mathcal{A}$ and $\mathtt{S}$ to $\mathcal{A}$.
3. For $1 \le i \le q$ for each pair $(\mathtt{Ow}, \mathtt{C})$ if $i \le q^{\mathtt{Ow},\mathtt{C}}$, $\mathcal{A}$ sends $\mathsf{w}_i^{\mathtt{Ow},\mathtt{C}}$, $\mathtt{Plcy}_i^{\mathtt{Ow},\mathtt{C}}$, and $\mathtt{Shr}_i^{\mathtt{Ow},\mathtt{C}}$.
$$\begin{pmatrix} \perp \\ K^c \end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}} \begin{pmatrix} \mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{Plcy}_i^{\mathtt{Ow},\mathtt{C}} \\ K^{\mathtt{C}}, \mathtt{Shr}_i^{\mathtt{Ow},\mathtt{C}} \end{pmatrix}$$
$$\begin{pmatrix} \perp \\ \mathsf{Res}_i^{\mathtt{Ow},\mathtt{C}} \end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}} \begin{pmatrix} \mathtt{EMM}^{\mathtt{Ow}} \\ K^{\mathtt{C}}, \mathsf{w}_i^{\mathtt{Ow},\mathtt{C}} \end{pmatrix}$$
In the above, the first $K^{\mathtt{C}} = \perp$., Send the view of parties in $\mathtt{Owners}_\mathcal{A} \cup \mathtt{Clients}_\mathcal{A}$ and $\mathtt{S}$ to $\mathcal{A}$.

$\mathcal{A}$ outputs a bit b.

$\underline{\mathsf{Exp}_{\mathcal{A},\mathsf{Sim},\mathsf{MKSE}}(1^\lambda, q)}$:

1. $\mathcal{A}$ sends to $\mathcal{C}$ all items as in Steps 1 to 4 of $\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{MKSE}}$.
2. $\mathsf{View}_0 \leftarrow \mathsf{Sim}(\mathcal{L}^{\mathsf{Setup}}(\mathtt{Owners}, \mathtt{Clients}, \mathtt{Ow}_\mathcal{A}, \mathtt{C}_\mathcal{A}, \vec{\mathsf{KOwn}}, \vec{\mathtt{MM}}))$ where
$$\vec{\mathsf{KOwn}} = \{(\mathtt{Ow}, \mathsf{KOwn})\}_{\mathtt{Ow} \in \mathtt{Owners}_\mathcal{A}}, \vec{\mathtt{MM}} = \{(\mathtt{Ow}, \mathtt{MM}^{\mathtt{Ow}})\}$$
3. For $1 \le i \le q$ for each pair $(\mathtt{Ow}, \mathtt{C})$ if $i \le q^{\mathtt{Ow},\mathtt{C}}$, $\mathcal{A}$ sends $\mathsf{w}_i^{\mathtt{Ow},\mathtt{C}}$, $\mathtt{Plcy}_i^{\mathtt{Ow},\mathtt{C}}$, and $\mathtt{Shr}_i^{\mathtt{Ow},\mathtt{C}}$.
$$\mathsf{View}_i \leftarrow \mathsf{Sim}(\mathcal{L}^{\mathsf{Grant}}(\vec{\mathtt{Plcy}}, \vec{\mathtt{Shr}}), \mathcal{L}^{\mathsf{Search}}(\vec{\mathsf{w}}))$$
where $\vec{\mathtt{Plcy}} = (\mathtt{Ow}, \mathtt{C}, \mathtt{Plcy}_1^{\mathtt{Ow},\mathtt{C}}, ..., \mathtt{Plcy}_i^{\mathtt{Ow},\mathtt{C}})$, $\vec{\mathsf{w}} = (\mathtt{Ow}, \mathtt{C}, \mathsf{w}_1^{\mathtt{Ow},\mathtt{C}}, ..., \mathsf{w}_i^{\mathtt{Ow},\mathtt{C}})$, $\vec{\mathtt{Shr}} = (\mathtt{Ow}, \mathtt{C}, \mathtt{Shr}_1^{\mathtt{Ow},\mathtt{C}}, ..., \mathtt{Shr}_i^{\mathtt{Ow},\mathtt{C}})$. Send the View_i to $\mathcal{A}$
4. Output $\mathcal{A}$'s output.

Figure 2: Security Game for MKSE.

a static dataset. This limitation is due to our use of PIR. While there are developments in incremental PIR [MZRA22] current solutions use multiple servers. Even with an updateable PIR, a solution is not obvious; keeping a DIP consistent requires managing evictions in a low-leakage way. We also consider arbitrary corruption of parties including owners, clients, and the server.

2.2 Technical Tools

Definition 2 (Data-Independent Packing [BBF+21])**.** *Let* $\mathtt{Map} : \mathcal{W} \to \{0,1\}^{\eta}$*. A* DIP *scheme consists of three algorithms* $(\mathsf{Size}, \mathsf{Build}, \mathsf{Lookup})$ *as follows,*

- $(\mu, \sigma, K) \leftarrow \mathsf{Size}(\ell)$*. Takes the number of values in the* Map *and returns the number of buckets* μ*, size of the stash* σ*, and the key* K *to evaluate the possible location of keywords.*
- $(\mathsf{Buckets}, \mathsf{Stash}) \leftarrow \mathsf{Build}(\mathtt{Map}, K)$*. Takes a map* Map *of size* ℓ *as input and outputs the pair* $(\mathsf{Buckets}, \mathsf{Stash})$*, where* $\mathsf{Buckets} = (\mathsf{Buckets}[1], ..., \mathsf{Buckets}[\mu])$ *and each* $\mathsf{Buckets}[i]$ *is a set of at most* ρ/η *map values.*
- $\mathsf{Pos}_{\mathsf{Buckets}} \leftarrow \mathsf{Lookup}(\mu, K, \mathsf{w})$*. Takes the total number of buckets* μ*, the DIP key* K*, a keyword* w*, and returns a set of bucket indices* $\mathsf{Pos}_{\mathsf{Buckets}} \subseteq [1, \mu]$*.*

Correctness *A* DIP *is correct if for all* Map *of size* ℓ*, it holds that:*

$$\Pr_{(\mu,\sigma,K)\leftarrow\mathsf{Size}(|\mathtt{Map}|)}\left[\forall \mathsf{w} \in \mathtt{Map} : \mathtt{Map}[\mathsf{w}] \subseteq \left(\mathsf{Stash} \cup \bigcup_{j\in\mathsf{Lookup}(\mu,K,\mathsf{w})} \mathsf{Buckets}[j]\right)\right] = 1.$$

Notation We abuse the notation using $\mathsf{Buckets}[\mathsf{w}]$ to denote all buckets that contain a map value that was associated with w. We will execute a DIP with keywords of the form (w, i) for an integer i. When we do this, we use

$$\mathsf{Buckets}[\mathsf{w}] := \cup_i \mathsf{Buckets}[(\mathsf{w}, i)],$$
$$\mathsf{Lookup}(\mu, K, \mathsf{w}, \mathtt{Vol}) := \cup_{i=1}^{\mathtt{Vol}} \mathsf{Lookup}(\mu, K, (\mathsf{w}, i))$$

as additional notations. For a DIP we use π_{DIP} to denote the implied permutation between $\mathsf{Arr}_{\mathtt{Map}}||(\perp, \perp)^{\mu\cdot\rho+\sigma-\ell}$ and $\mathsf{Buckets}||\mathsf{Stash}$. This permutation is a function of the randomness of Build, K and Map. For a Map that is transformed using a PRF before building the DIP, we use $\mathsf{Buckets}[F(\cdot, \mathsf{w})]$ to indicate the locations of w in the DIP, in this notation the relevant key for the PRF will be clear from context.

When we define a multimap MM, all documents have size η. When creating ValMap, this would yield a map of size $\ell \sum_{\mathsf{w}} |\mathtt{MM}[\mathsf{w}]|\eta$. The buckets can fit ρ/η values, so one obtains a map of size approximately $\ell \sum_{\mathsf{w}} \lceil \frac{\eta|\mathtt{MM}[\mathsf{w}]|}{\rho} \rceil$. Notationally, we ignore the task of packing ρ/η together, but this is handled by our implementation.

Intuition For Construction Previous DIPs are based on ideas from Cuckoo hashing [Yeo23]. The Size algorithm chooses two hash functions and a number of buckets. Based on these hash functions, each keyword can go in one of the two buckets or the stash. Build is responsible for organizing the actual location of the items to fit between the possible buckets and the stash. TethysDIP [BBF+21] showed that the minimum size of the stash was equal to the max-flow of a graph with edges being keywords and nodes being the set of possible buckets. We slightly abuse their construction which was proved optimal when $\mu = 2\ell + \Theta(1)$, we consider $\mu < 2\ell$ in our analysis, see Section 5.

Definition 3. *(PIR with preprocessing [CHR17, LMW23, CD24]) Let* ρ *be some positive integer and define* $\Sigma = \{0,1\}^{\rho}$*. A PIR (with preprocessing) is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Query}, \mathsf{Resp}, \mathsf{Rec})$ *acting on a size-*μ *database with the following syntax:*

- $\tilde{\mathcal{DB}} \leftarrow \mathsf{Setup}(\mathcal{DB})$*. Takes a database* $\mathcal{DB}$ *over* $\{0,1\}^{\mu\times\rho}$ *for alphabet* $\{0,1\}^{\rho}$ *and samples a preprocessed database* $\tilde{\mathcal{DB}}$ *to be handed to the server.*

- $(\mathsf{q}, \mathsf{st}) \leftarrow \mathsf{Query}(i)$. *Takes as input an address* $i \in \{1, \cdots, \mu\}$*, and returns a query* q *and a local state* st*.*
- $\mathsf{ans} \leftarrow \mathsf{Resp}(\tilde{\mathcal{DB}}, \mathsf{q})$. *Returns the server's response* ans *given the processed database* $\tilde{DB}$ *and query* q*.*
- $\mathcal{DB}'_i \leftarrow \mathsf{Rec}(\mathsf{st}, \mathsf{q}, \mathsf{ans})$. *Returns the decoded value.*

Correctness *Fix some* $\mathcal{DB} \in \Sigma^\mu$*. We say that the PIR is correct if for all* $i \in [1, \mu]$*,*

$$\Pr\left[\mathcal{DB}'_i = \mathcal{DB}[i] \;\middle|\; \begin{array}{r} \tilde{\mathcal{DB}} \leftarrow \mathsf{Setup}(\mathcal{DB}) \\ (\mathsf{q}, \mathsf{st}) \leftarrow \mathsf{Query}(i) \\ \mathsf{ans} \leftarrow \mathsf{Resp}(\tilde{\mathcal{DB}}, \mathsf{q}) \\ \mathcal{DB}'_i \leftarrow \mathsf{Rec}(\mathsf{st}, \mathsf{q}, \mathsf{ans}) \end{array}\right] > 1 - \mathsf{negl}(\lambda).$$

Security *For a PPT algorithm* $\mathcal{A}$*, define* $\mathtt{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{PIR}}(1^\lambda, q)$ *as:*
$\underline{\mathtt{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{PIR}}(1^\lambda, q)}$

1. $b \xleftarrow{\$} \{0, 1\}$
2. $\mathsf{pp}_{\mathcal{DB}} \leftarrow \mathsf{PIR.Setup}(1^\kappa)$
3. $(i_1^0, ..., i_q^0), (i_1^1, ..., i_q^1) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathcal{DB}}, \mathcal{DB})$
4. *For* $\iota = 1$ *to* q*:*
   (a) *Define* $t_\iota = i_\iota^0 + b * (i_\iota^1 - i_\iota^0)$.
   (b) *Define* $(\mathsf{q}_\iota, \mathsf{st}) \leftarrow \mathsf{Query}(\mathsf{pp}_{\mathcal{DB}}, t_\iota)$
5. $b' \leftarrow \mathcal{A}(\mathsf{pp}_{\mathcal{DB}}, \mathcal{DB}, \vec{\mathsf{q}})$
6. *Output* $b' == b$.

We say that PIR *is secure if for all PPT* $\mathcal{A}$ *and all* $q = \mathtt{poly}(\lambda)$*,*

$$|\Pr[\mathtt{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{PIR}}(1^\lambda, q)] - 1/2| < \mathsf{negl}(\lambda).$$

Since there are no state updates, single query correctness implies multi query correctness for a polynomial number of queries.

Shift Friendly *We additionally say that a* PIR *is shift friendly if there is an additional pair of algorithms* $\mathsf{Shift}, \mathsf{Rec}_\mathsf{S}$ *such that* $\forall i, j \in [1, \mu]$*, define* $k := i + j \mod \mu$*,*

$$\Pr\left[\mathcal{DB}' = \mathcal{DB}[k] \;\middle|\; \begin{array}{r} \tilde{\mathcal{DB}} \leftarrow \mathsf{Setup}(\mathcal{DB}) \\ (\mathsf{q}, \mathsf{st}) \leftarrow \mathsf{Query}(i) \\ (\mathsf{q}', \mathsf{st}) \leftarrow \mathsf{Shift}(j, \mathsf{q}) \\ \mathsf{ans} \leftarrow \mathsf{Resp}(\tilde{\mathcal{DB}}, \mathsf{q}') \\ \mathcal{DB}' \leftarrow \mathsf{Rec}_\mathsf{S}(\mathsf{st}, \mathsf{q}, j, \mathsf{ans}) \end{array}\right] > 1 - \mathsf{negl}(\lambda).$$

As we discuss in Section 3.2.1, circular right shift exists for both DEPIR and FrodoPIR. We use this property to condense our MKSE into a single round in Section 3.

2.3 Access Control

We now introduce our last technical tool that we call an *access control* mechanism. Roughly, an *access control* mechanism is an encryption of an array (corresponding to a $\texttt{Map}$) that does not reveal anything about the contents of the map beyond what is revealed by decryptions available to the adversary. This is true even with the adversary chooses the order of the ciphertexts by choosing a permutation to be applied to the plaintexts. The object can be viewed as a version of real-or-random security for encryption where there are multiple keys.

Definition 4. *Let* $\mathsf{AcCtrl} = (\mathsf{Setup}, \mathsf{Grant}, \mathsf{Decrypt})$ *where* Setup *is a randomized function,* Grant *is a protocol (between* $\texttt{Ow}$ *and* $\texttt{C}$*), and* $\mathsf{Decrypt}$ *is a deterministic function. Let* η, ℓ *be integers and* $\mathcal{W}, \mathcal{K}$ *be finite sets representing the domain of keywords and keys respectively. Let*

$$(\mathsf{CArr}_{\mathsf{EDocs}}, \mathsf{KOwn}) \leftarrow \mathsf{Setup}^{\texttt{Ow}}(\mathsf{Arr}_{\texttt{Map}})$$

be a function that takes as input an array, of size ℓ*, corresponding to a map* $\texttt{Map}$*. The return value is* $\mathsf{CArr}_{\mathsf{EDocs}}$*, which is* $\mathsf{Arr}_{\texttt{Map}}$ *except that each* Doc *is replaced with a ciphertext. We use* $\mathsf{CArr}_{\mathsf{EDocs}}[\mathsf{w}]$ *to mean the array position of keyword* w*.* $\mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}}$ *is* $\mathsf{CArr}_{\mathsf{EDocs}}$ *with all keywords removed. We similarly define* $\mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}}[\mathsf{w}]$ *note that this is not necessarily efficiently computable from the array itself.* KOwn *is a key.*

The protocol

$$\begin{pmatrix} \perp \\ \vec{K_{\mathsf{w}}} \end{pmatrix} \leftarrow \mathsf{Grant}^{\texttt{Ow},\texttt{C}} \begin{pmatrix} \mathsf{KOwn}, \texttt{Plcy} \\ \vec{\mathsf{w}} \end{pmatrix}$$

retrieves the keys corresponding to $\vec{\mathsf{w}}$ *as long as the policy is satisfied. Then* $\vec{\mathsf{Doc}} = \mathsf{Decrypt}^{\texttt{C}}(\vec{c}, \vec{K_{\mathsf{w}}})$ *where* $\vec{c}$ *is some arbitrary subset of* $\mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}}$*. A* AcCtrl *is a good access control mechanism for leakage functions* $(\mathcal{L}^{\mathsf{Setup}}, \mathcal{L}^{\mathsf{Grant}}_{\texttt{Ow}}, \mathcal{L}^{\mathsf{Grant}}_{\texttt{C}})$ *if for security parameter* λ *and all PPT* $\mathcal{A}$ *for experiments defined as in Figure 3 it is true that*

1. ***Correctness*** *Define*

   $\underline{\mathtt{Exp}_{\mathtt{Correct},\mathcal{A},\mathsf{AcCtrl}}(1^\lambda, q)}$

   $\mathcal{A}$ *sends an array* $\mathsf{Arr}_{\texttt{Map}}$*,* $\texttt{Plcy}_1, ..., \texttt{Plcy}_t$*, and queries* $\mathsf{w}_1, ..., \mathsf{w}_q$*.*

   $\mathcal{C}$ *performs:*

   (a) $(\mathsf{CArr}_{\mathsf{EDocs}}, \mathsf{KOwn}) \leftarrow \mathsf{Setup}(\mathsf{Arr}_{\texttt{Map}})$.

   (b) *For* $i = 1$ *to* q*,*

      i. *Define* $\vec{c_i} = \mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}}[\mathsf{w}_i]$.

      ii. *Define*

      $$\begin{pmatrix} \perp \\ \vec{K_{i,\mathsf{w}}} \end{pmatrix} \leftarrow \mathsf{Grant}^{\texttt{Ow},\texttt{C}} \begin{pmatrix} \mathsf{KOwn}, \texttt{Plcy}_i \\ \mathsf{w}_i \end{pmatrix}$$
      $$\vec{\mathsf{Docs}}_i = \mathsf{Decrypt}^{\texttt{C}}(\vec{c_i}, \vec{K_{i,\mathsf{w}}}).$$

   (c) *Output* $\forall i, s.t. \texttt{Plcy}_i(\mathsf{w}_i) = 1, \vec{\mathsf{Docs}}_i \supseteq \mathsf{Arr}[\mathsf{w}_i]$.

   Then it is true that for all PPT $\mathcal{A}$ *there exists some negligible function* $\texttt{ngl}$ *such that* $\Pr[\mathtt{Exp}_{\mathtt{Correct},\mathcal{A},\mathsf{AcCtrl}}(1^\lambda, q) = 1] \geq 1 - \texttt{ngl}(\lambda)$.

2. ***Security*** *For every PPT* $\mathcal{A}$*, there exists some PPT* Sim *such that*

$$\left| \begin{matrix} \Pr\left[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{AcCtrl}}(1^\lambda, q, \ell)\right] \\ -\Pr[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{Sim},\mathsf{AcCtrl}}(1^\lambda, q, \ell, \mathcal{L}_{\texttt{Ow}}, \mathcal{L}_{\texttt{C}}) = 1] \end{matrix} \right| \leq \texttt{ngl}(\lambda).$$

$\underline{\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{AcCtrl}}(1^\lambda, q, \ell)}$

$\mathcal{A}$ sends to $\mathcal{C}$: $\mathsf{ShrSetup} = \mathsf{Arr}, \pi, 1_{\mathtt{Ow},\mathcal{A}}, 1_{\mathtt{C},\mathcal{A}}$ where Arr of size ℓ, a permutation $\pi : [1,\ell] \to [1,\ell]$, and two bits $1_{\mathtt{Ow},\mathcal{A}}, 1_{\mathtt{C},\mathcal{A}}$.

1. π specifies the output positions of documents.
2. The bits $1_{\mathtt{Ow},\mathcal{A}}, 1_{\mathtt{C},\mathcal{A}}$ specifying adversarial behavior, constrained that $1_{\mathtt{Ow},\mathcal{A}} + 1_{\mathtt{C},\mathcal{A}} \leq 1$. If not, output 0.

$\mathcal{C}$ performs:

1. $(\mathsf{CArr}_{\mathsf{EDocs}}, \mathsf{KOwn}) \leftarrow \mathsf{Setup}^{\mathtt{Ow}}(\mathsf{Arr}_{\mathtt{Map}})$. Send $\pi(\mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}})$ to $\mathcal{A}$ and view of $\mathtt{Ow}$ if $1_{\mathtt{Ow},\mathcal{A}}$.
2. For $i = 1$ to q,
   (a) $\mathcal{A}$ sends $\mathtt{Plcy}_i : \mathcal{W} \to \{0,1\}$, $\mathsf{Pos}_i \subseteq [1,\ell]$ the set of positions to be revealed and w_i.
   (b) Execute
   $$\begin{pmatrix} \perp \\ \vec{\mathsf{Docs}}_i \end{pmatrix} \leftarrow \mathsf{Grant}^{\mathtt{Ow},\mathtt{C}} \begin{pmatrix} \mathsf{KOwn}, \mathtt{Plcy}_i \\ \pi(\mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}})[\mathsf{w}_i], \mathsf{w}_i \end{pmatrix}$$
   (c) Provide view of adversarial party to $\mathcal{A}$.

$\mathcal{A}$ outputs a bit b.

$\underline{\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{Sim},\mathsf{AcCtrl}}(1^\lambda, q, \ell, \mathcal{L}_{\mathtt{Ow}}, \mathcal{L}_{\mathtt{C}})}$

$\mathcal{A}$ sends $\mathsf{ShrSetup}$ to $\mathcal{C}$ as in $\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{AcCtrl}}(1^\lambda, q, \ell)$. Output 0 if $1_{\mathtt{Ow},\mathcal{A}} + 1_{\mathtt{C},\mathcal{A}} = 2$.
If $1_{\mathtt{Ow},\mathcal{A}} = 1$, $\mathtt{View}_0 \leftarrow \mathsf{Sim}(\mathsf{ShrSetup})$ else $\mathtt{View}_0 \leftarrow \mathsf{Sim}(\mathcal{L}^{\mathsf{Setup}}(\mathsf{ShrSetup}))$.
For $i = 1$ to q perform

1. $\mathcal{A}$ sends $\mathsf{ShrGrant}_i = (\mathsf{w}_i, \mathsf{Pos}_i, \mathtt{Plcy}_i)$ to $\mathcal{C}$.
2. **If** $1_{\mathtt{C},\mathcal{A}}$: $\mathtt{View}_i \leftarrow \mathsf{Sim}(\mathsf{w}_i, \mathsf{Pos}_i, \mathcal{L}^{\mathsf{Grant}}_{\mathtt{C}}(\mathsf{ShrSetup}, \mathsf{ShrGrant}))$.
3. **Else if** $1_{\mathtt{Ow},\mathcal{A}}$, $\mathtt{View}_i \leftarrow \mathsf{Sim}(\mathtt{Plcy}_i, \mathcal{L}^{\mathsf{Grant}}_{\mathtt{Ow}}(\mathsf{ShrSetup}, \mathsf{ShrGrant}_i))$.
4. **Else** $\mathtt{View}_i \leftarrow \mathsf{Sim}(1^\lambda)$.
5. Send $\mathtt{View}_i$ to $\mathcal{A}$.

$\mathcal{A}$ outputs a bit b.

Figure 3: Security Experiments for AcCtrl.

3 Overall MKSE Design

We provide an overview of how MKSE schemes are usually constructed (including in MARS). For input MM, one creates two maps that we call VolMap and ValMap.

Construction 1 (MKSE from VolMap and ValMap)**.** *Let* $\mathsf{MKSE}_{\mathtt{VolMap}}$ *and* $\mathsf{MKSE}_{\mathtt{ValMap}}$ *be* MKSE *schemes for maps with leakage functions* $\mathcal{L}_{\mathtt{VolMap}}$ *and* $\mathcal{L}_{\mathtt{ValMap}}$*. Then define* $\mathsf{MKSE}_{\mathtt{MM}}$ *for* MM *as in Figure 4.*

Theorem 1. *The construction of* $\mathsf{MKSE}_{\mathtt{MM}}$ *from Construction 1 is a good* MKSE *with leakage functions* $\mathcal{L}_{\mathtt{MM}} = \mathcal{L}_{\mathtt{VolMap}} \cup \mathcal{L}_{\mathtt{ValMap}} \cup_{\mathsf{w} \in \mathsf{Query}} |\mathtt{MM}[\mathsf{w}]|$*. In the above, for each* Share, Search *invocation, there are* $\mathtt{Vol} = |\mathtt{MM}[\mathsf{w}]|$ *calls to the relevant algorithm for* ValMap*, this reveals* Vol *in addition to leakage from* ValMap*.*

We do not prove Theorem 1, the proof is straightforward.

On use of Owner sending $\perp$ In Figure 4, in Share the client sends requests to the $\mathsf{Share}_{\mathtt{ValMap}}$ protocol until they are told to stop. In our main construction, the Ow learns w so it can compute the relevant pairs (w, i) from a single request. This protocol supports single round Share without requiring a stop message from Ow.

Next we provide alternatives for an MKSE encryption of a Map. We then discuss our preferred combination of these techniques in Construction 1 for VolMap and ValMap to yield an MKSE of an MM.

$$\underline{\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}\\ \mathtt{EMM}\end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{MM}\\ \perp\end{pmatrix}}$$

$\mathtt{Ow}$ does:

1. Parse $\mathsf{KOwn}^{\mathtt{Ow}}$ as $\mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{VolMap}}, \mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{ValMap}}$.
2. Define $\mathtt{Map}_{\mathtt{VolMap}}, \mathtt{Map}_{\mathtt{ValMap}}$ as empty maps.
3. For each $\mathsf{w} \in \mathtt{MM}$,
   (a) Define $\mathsf{Arr}[\mathsf{w}]$ and $\mathtt{Map}_{\mathtt{VolMap}}[\mathsf{w}] = |\mathtt{MM}[\mathsf{w}]|$.
   (b) For $i = 1$ to $|\mathsf{Arr}[\mathsf{w}]|$, set $\mathtt{Map}_{\mathtt{ValMap}}[(\mathsf{w}, i)] = \mathsf{Arr}[\mathsf{w}][i]$
4. $\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{VolMap}}\\ \mathtt{EMap}_{\mathtt{VolMap}}\end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{Map}_{\mathtt{VolMap}}\\ \perp\end{pmatrix}$
5. $\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{ValMap}}\\ \mathtt{EMap}_{\mathtt{ValMap}}\end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{Map}_{\mathtt{ValMap}}\\ \perp\end{pmatrix}$

$$\underline{\begin{pmatrix}\perp\\ \mathsf{Res}\end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}}\begin{pmatrix}\mathtt{EMM}\\ K^{\mathtt{C}}, \mathsf{w}\end{pmatrix}}$$

1. Parse K^C as $K^C_{\mathtt{VolMap}}, K^C_{\mathtt{ValMap}}$.
2. Parse $\mathtt{EMM}$ as $\mathtt{EMap}_{\mathtt{VolMap}}, \mathtt{EMap}_{\mathtt{ValMap}}$.
3. Perform $\begin{pmatrix}\perp\\ \mathtt{Vol}\end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}}\begin{pmatrix}\mathtt{EMap}_{\mathtt{VolMap}}\\ K^{\mathtt{C}}_{\mathtt{VolMap}}, \mathsf{w}\end{pmatrix}$

$$\forall 1 \le i \le \mathtt{Vol}, \begin{pmatrix}\perp\\ \mathsf{Res}_i\end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}}\begin{pmatrix}\mathtt{EMap}_{\mathtt{ValMap}}\\ K^{\mathtt{C}}_{\mathtt{ValMap}}, (\mathsf{w}, i)\end{pmatrix}$$

4. Output $\mathsf{Res} = \mathsf{Res}_1, ..., \mathsf{Res}_{\mathtt{Vol}}$.

$$\underline{\begin{pmatrix}\perp\\ K^{\mathtt{C}}\end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}, \mathtt{Plcy}\\ K^{\mathtt{C}}, \mathsf{w}\end{pmatrix}}$$

1. Parse K^C as $K^C_{\mathtt{VolMap}}, K^C_{\mathtt{ValMap}}$.
2. Parse $\mathtt{EMM}$ as $\mathtt{EMap}_{\mathtt{VolMap}}, \mathtt{EMap}_{\mathtt{ValMap}}$.
3. Parse $\mathsf{KOwn}^{\mathtt{Ow}}$ as $\mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{VolMap}}, \mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{ValMap}}$.
4. $\begin{pmatrix}\perp\\ K^{\mathtt{C}}_{\mathtt{VolMap}}\end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{VolMap}}, \mathtt{Plcy}\\ K^{\mathtt{C}}_{\mathtt{VolMap}}, \mathsf{w}\end{pmatrix}$
5. Until $\mathtt{Ow}$ sends $\perp$ starting at $i = 1$
   (a) $\begin{pmatrix}\perp\\ K^{\mathtt{C}}_{\mathtt{VolMap}}\end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}_{\mathtt{ValMap}}, \mathtt{Plcy}\\ K^{\mathtt{C}}_{\mathtt{ValMap}}, (\mathsf{w}, i)\end{pmatrix}$
   (b) Set $i = i + 1$

Figure 4: MKSE for $\mathtt{MM}$ from MKSE for $\mathtt{Map}$.

3.1 KPIR: Low-Leakage MKSE for Map

In this subsection, we show how to achieve a low-leakage, MKSE for a $\mathtt{Map}$ from a:

$$\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}\\ \mathtt{EMap}\end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}}\begin{pmatrix}\mathsf{KOwn}^{\mathtt{Ow}}=\perp, \mathtt{Map}\\ \perp\end{pmatrix}$$

$\mathtt{Ow}$ does:

1. Sample $\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}} \leftarrow \{0,1\}^\lambda$.
2. For each w define $\mathtt{Map}_{\mathsf{PRF}}[F(\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}}, \mathsf{w})] = \mathtt{Map}[\mathsf{w}]$
3. Run $(\mu_{\mathsf{EDocs}}, \sigma_{\mathsf{EDocs}}, K_{\mathsf{DIP},\mathsf{EDocs}}) \leftarrow \mathsf{DIP.Size}(|\mathtt{Map}_{\mathsf{PRF}}|)$.
4. Define $\mathsf{Arr} = \mathsf{Arr}_{\mathtt{Map}_{\mathsf{PRF}}}$.
5. Append $0^{(\mu_{\mathsf{EDocs}}\cdot p+\sigma_{\mathsf{EDocs}}-\ell)\cdot\eta}$ to Arr.
6. $(\mathsf{CArr}_{\mathsf{EDocs}}, \mathsf{KOwn}_{\mathsf{AcCtrl}}) \leftarrow \mathsf{AcCtrl.Setup}^{\mathtt{Ow}}(\mathsf{Arr})$.
7. Define $\mathsf{CArr}^{\mathsf{No\ w}}_{\mathsf{EDocs}}$ to be $\mathsf{CArr}_{\mathsf{EDocs}}$ with keywords removed.
8. Set $\mathsf{KOwn}^{\mathtt{Ow}} = \mathsf{KOwn}_{\mathsf{AcCtrl}}||\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}}$.
9. $(\mathsf{Buckets}, \mathsf{Stash}) \leftarrow \mathsf{DIP.Build}(\mathtt{Map}_{\mathsf{PRF}}, K_{\mathsf{DIP},\mathsf{EDocs}})$.
10. Let π_{DIP} the implied permutation from $\mathsf{Buckets}$ and define $\mathsf{CBuck} := \pi_{\mathsf{DIP}}(\mathsf{CArr}_{\mathsf{EDocs}})$.
11. Send $(\mathsf{CBuck}, \mathsf{Stash}, \mu_{\mathsf{EDocs}}, \sigma_{\mathsf{EDocs}}, K_{\mathsf{DIP},\mathsf{EDocs}})$ to $\mathtt{S}$.

$\mathtt{S}$ receives $(\mathsf{CBuck}, \mathsf{Stash}, \mu_{\mathsf{EDocs}}, \sigma_{\mathsf{EDocs}}, K_{\mathsf{DIP},\mathsf{EDocs}})$ and:

1. Runs $\mathtt{EMap}_{\mathsf{EDocs}} \leftarrow \mathsf{PIR.Setup}(\mathsf{CBuck})$.
2. $\mathtt{EMap} := \mathtt{EMap}_{\mathsf{EDocs}}, \mathsf{Stash}_{\mathsf{EDocs}}, \mu_{\mathsf{EDocs}}, \sigma_{\mathsf{EDocs}}, K_{\mathsf{DIP},\mathsf{EDocs}}$.

Figure 5: $\mathtt{KPIR}$ Setup construction.

1. DIP organizes the $\mathtt{Map}$ into a fixed size array with a public $\mathsf{Translate}$ function combined with a private PRF,
2. PIR allows private search, and
3. AcCtrl shares the relevant keys (and allows decryption).

We defer introducing constructions of access control mechanism until Section 4.

Construction 2 (MKSE from DEPIR and AcCtrl)**.** *Let* $\mathtt{Map}$ *be a map of size* ℓ *and let:*

1. F *be a pseudorandom function from* $\{0,1\}^\lambda \times \mathcal{W}$ *to* $\{0,1\}^\lambda$,
2. DIP *be a Data Independent Packing on maps* $\mathtt{Map} : \{0,1\}^\lambda \to \{0,1\}^\eta$ *that outputs buckets each of size* ρ,
3. PIR *over* $\Sigma = \{0,1\}^\rho$ *be a PIR scheme, and*
4. $\mathsf{AcCtrl} = (\mathsf{Setup}_{\mathsf{AcCtrl}}, \mathsf{Grant}_{\mathsf{AcCtrl}})$ *be an access control mechanism.*

Define $\mathsf{KeyGen}^{\mathtt{Ow}} = \perp$. Setup *is defined in Figure 5 and* Search *is defined in Figure 6.*

$$\begin{pmatrix}\perp\\ \mathsf{Res}\end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}_{\mathsf{AcCtrl}}, \mathtt{Plcy}\\ \mathsf{CRes}, \mathsf{w}\end{pmatrix}$$

1. *Run* $\begin{pmatrix}\perp\\ \mathsf{Res}\end{pmatrix} \leftarrow \mathsf{AcCtrl.Grant}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}_{\mathsf{AcCtrl}}, \mathtt{Plcy}\\ \mathsf{CRes}, \mathsf{w}\end{pmatrix}$
2. $\mathtt{C}$ *sends* w *and* $\mathtt{Ow}$ *responds with* $F(\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}}, \mathsf{w})$.
3. $\mathtt{C}$ *stores* $\mathsf{LocMap}[\mathsf{w}] = F(\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}}, \mathsf{w})$

$$\begin{pmatrix} \perp \\ \mathsf{CRes} \end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}} \begin{pmatrix} \mathtt{EMap} \\ K^{\mathtt{C}'}, \mathsf{w} \end{pmatrix}$$

$\mathtt{C}$ does:

1. $\mathsf{Pos}_{\mathsf{Buckets}} = \mathsf{DIP.Lookup}(\mu_{\mathsf{EDocs}}, \mathsf{K}_{\mathsf{DIP,EDocs}}, \mathsf{LocMap}[\mathsf{w}])$.
2. For each $i \in \mathsf{Pos}_{\mathsf{Buckets}}$, run:
   (a) $\mathsf{q}_i, \mathsf{st}_i = \mathsf{PIR}_{\mathsf{EDocs}}.\mathsf{Query}(i)$.
   (b) Send q_i to $\mathtt{S}$.

$\mathtt{S}$ for each q_i, send $\mathsf{ans} = \mathsf{PIR}_{\mathsf{EDocs}}.\mathsf{Resp}(\mathtt{EMM}_{\mathsf{EDocs}}, \mathsf{q})$ to $\mathtt{C}$. Also send $\mathsf{Stash}_{\mathsf{EDocs}}$ to $\mathtt{C}$.

$\mathtt{C}$ does

1. $\mathsf{CRes} = \mathsf{CRes}||\mathsf{PIR}_{\mathsf{EDocs}}.\mathsf{Rec}(\mathsf{st}, \mathsf{ans})$ for each ans.
2. For $\mathsf{Doc} \in \mathsf{Stash}_{\mathsf{EDocs}}$, $\mathsf{CRes} = \mathsf{CRes}||\mathsf{Doc}$.

Figure 6: KPIR Search protocol.

Setup. As shown in Figure 5, first the owner samples a PRF key and applies a PRF evaluation to all the existing keywords in the input Map. As described in the Introduction, this lets the owner to define a new map called $\mathsf{M}_{\mathsf{PRF}}$ that is indexed by PRF evaluation of keywords, instead of keywords themselves. The owner passes the size of this map to the DIP.Size() to receive the buckets' sizes and the key to the DIP. In line 4, the owner initializes a new array Arr, to store the elements of $\mathsf{M}_{\mathsf{PRF}}$. In the next step, the owner calls the preferred sharing mechanism's Setup over this array to receive an encrypted array $\mathsf{Arr}_{\mathsf{EDocs}}$ along with the sharing key. The sharing mechanism is explained in Section 4. In line 9, the owner performs the packing by calling $\mathsf{DIP.Build}(\mathtt{Map}_{\mathsf{PRF}})$; after that in line 10, the owner uses the permutation implied by applying the DIP to permute the output ciphertexts from sharing mechanism and passes them to the Server. The Server $\mathtt{S}$, runs PIR.Setup() over the received array and stores the output along with the Stash.

Search. As shown in Figure 6, for any queried keyword w, the Client $\mathtt{C}$ first needs to retrieve the PRF evaluation of w to learn the keyword to input to the DIP.Lookup. Then for each position from the DIP, the client prepares a PIR query and sends it to the server. As the server replies with an answer, the client reconstructs the result.

Notes In the above protocol, $\mathtt{C}$ reveals w to learn the positions to search. We present the construction this was as our default AcCtrl reveals w to malicious $\mathtt{Ow}$. As we discuss in Section 4.1.1 one can modify AcCtrl to use an OPRF to not reveal w of course, one can use a OPRF to communicate outputs of $F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \cdot)$. Since all parties are semi-honest one does not need to enforce consistency between the two OPRF inputs. Both of these approaches can be done with Share having the same number of rounds as Grant.

Theorem 2. *Let all algorithms be defined as in Construction 2 where* AcCtrl *has leakage* $(\mathcal{L}^{\mathsf{AcCtrl}}_{\mathtt{Ow}}, \mathcal{L}^{\mathsf{AcCtrl}}_{\mathtt{C}})$. *Then, Construction 2 is a secure MKSE scheme (Definition 1) with the leakage profile shown in Figure 7.*

Proof of Theorem 2. We separately consider correctness and security according to Definition 1.

Correctness. Correctness is not guaranteed if there exist two $\mathsf{w}_1, \mathsf{w}_2 \in \mathtt{Map}$ such that $F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \mathsf{w}_1) = F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \mathsf{w}_2)$ by security of the PRF, this occurs with negligible probability. Conditioned on this not occuring, correctness follows from the overwhelming correctness of AcCtrl, DIP, and PIR in sequence (Definitions 4, 2, and 3).

$\underline{\mathcal{L}^{\mathsf{Setup}}(\mathtt{Ow}_c, \mathtt{C}_c, \{\mathsf{KOwn}^i\}_{\mathtt{Ow}^i \in \mathtt{Ow}_c}, \mathtt{Map}) :}$

1. The number of clients, n, owners m.
2. For each $\mathtt{Ow}$, $\kappa^{\mathtt{Ow}}$ the number of keywords in the database.
3. The identities and keys (KOwn_i and K_j) for all users in $\mathtt{Ow}_{\mathcal{A}}, \mathtt{C}_{\mathcal{A}}$.
4. Receive $\forall \mathtt{Ow}, \mathtt{C}, q^{\mathtt{Ow},\mathtt{C}}$ (The number of queries).
5. If $\mathtt{Ow} \in \mathtt{Ow}_{\mathcal{A}}, \forall \mathtt{C}$, receive $\vec{\mathtt{Plcy}}^{\mathtt{Ow},\mathtt{C}}$ and receive $\mathtt{Map}^{\mathtt{Ow}}$.
6. Reveal $\mathcal{L}^{\mathsf{Setup}}_{\mathtt{KPIR}}$.

$\underline{\mathcal{L}^{\mathsf{Setup}}_{\mathtt{KPIR}}(\mathtt{Map}^{\mathtt{Ow}})}$:

1. Run $(\mu^{\mathtt{Ow}}, \sigma^{\mathtt{Ow}}, K^{\mathtt{Ow}}_{\mathsf{DIP}}) \leftarrow \mathsf{DIP}(|\mathtt{Map}^{\mathtt{Ow}}|)$.
2. Run $(\mathsf{Buckets}^{\mathtt{Ow}}, \mathsf{Stash}^{\mathtt{Ow}}) \leftarrow \mathsf{Build}(\mathtt{Map}^{\mathtt{Ow}})$.
3. Define $\mathsf{Arr}_{\mathsf{DIP}} = \mathsf{Buckets}^{\mathtt{Ow}} || \mathsf{Stash}^{\mathtt{Ow}}$ and $\pi^{\mathtt{Ow}}_{\mathsf{DIP}}$ as the implied permutation between $\mathsf{Arr}_{\mathtt{Map}^{\mathtt{Ow}}}$ and $\mathsf{Arr}_{\mathsf{DIP}}$.
4. Reveal $K^{\mathtt{Ow}}_{\mathsf{DIP}}, \mathsf{Arr}_{\mathtt{Map}^{\mathtt{Ow}}}, \pi$.
5. Reveal $\mathcal{L}^{\mathsf{AcCtrl},\mathsf{Setup}}(\mathsf{ShrSetup})$

$\underline{\mathcal{L}^{\mathsf{Search}}_{\mathtt{KPIR}}(\mathtt{Plcy}^{\mathtt{Ow},\mathtt{C}}, \mathsf{w}^{\mathtt{Ow},\mathtt{C}}, \mathtt{Shr}^{\mathtt{Ow},\mathtt{C}})}$:

1. Define $\mathsf{Pos}^{\mathtt{Ow},\mathtt{C}} = \mathsf{Arr}_{\mathtt{Map}^{\mathtt{Ow}}}[\mathsf{w}^{\mathtt{Ow},\mathtt{C}}]$. Note this includes all possible buckets and the entire stash for every $\mathsf{w}^{\mathtt{Ow},\mathtt{C}}_i$.
2. If $\mathtt{Ow} \in \mathtt{Ow}_{\mathcal{A}}$ and $\mathtt{C} \in \mathtt{C}_{\mathcal{A}}$ reveal $(\mathtt{Plcy}^{\mathtt{Ow},\mathtt{C}}, \mathsf{w}^{\mathtt{Ow},\mathtt{C}}, \mathtt{Shr}^{\mathtt{Ow},\mathtt{C}}, \mathsf{Pos}^{\mathtt{Ow},\mathtt{C}})$
3. If $\mathtt{Ow} \notin \mathtt{Ow}_{\mathcal{A}}$ and $\mathtt{C} \in \mathtt{C}_{\mathcal{A}}$, reveal $\mathsf{Pos}^{\mathtt{Ow},\mathtt{C}}$, and $\mathcal{L}^{\mathsf{AcCtrl},\mathsf{Grant}}_{\mathtt{C}}(\mathtt{Plcy}^{\mathtt{Ow},\mathtt{C}}, \mathsf{w}^{\mathtt{Ow},\mathtt{C}}, \mathsf{Pos}^{\mathtt{Ow},\mathtt{C}}, \mathtt{Shr}^{\mathtt{Ow},\mathtt{C}})$.
4. If $\mathtt{Ow} \in \mathtt{Ow}_{\mathcal{A}}$ and $\mathtt{C} \notin \mathtt{C}_{\mathcal{A}}$ reveal $\mathtt{Plcy}^{\mathtt{Ow},\mathtt{C}}$ and $\mathcal{L}^{\mathsf{AcCtrl}}_{\mathtt{Ow}}(\mathtt{Plcy}^{\mathtt{Ow},\mathtt{C}}, \mathsf{w}^{\mathtt{Ow},\mathtt{C}}, \mathsf{Pos}^{\mathtt{Ow},\mathtt{C}}, \pi^{\mathtt{Ow}}_{\mathsf{DIP}}, \mathtt{Shr}^{\mathtt{Ow}}) \cup \mathsf{w}^{\mathtt{Ow},\mathtt{C}}$.

Figure 7: Leakage profiles

Security. Fix some adversary $\mathcal{A}_{\mathsf{MKSE}}$. Our goal is to build a simulator $\mathsf{Sim}_{\mathsf{MKSE}}$ such that for all $q, t = \mathtt{poly}(\lambda)$:

$$\left| \begin{matrix} \Pr[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t) = 1] \\ -\Pr[\mathsf{Exp}_{\mathcal{A}_{\mathsf{MKSE}},\mathsf{Sim}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t) = 1] \end{matrix} \right| \leq \mathtt{ngl}(\lambda).$$

We consider mn intermediate experiments where the interactions between the experiment and the adversary up to the ith $\mathtt{Ow}$ and jth $\mathtt{C}$ are replaced by a simulated view. Notationally, we use $\mathsf{Exp}^{i,j}_{\mathcal{A}_{\mathsf{MKSE}},\mathsf{Sim}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t) = 1$. For the above value to non-negligible it must be the case that either $j < m$

$$\left| \begin{matrix} \Pr[\mathsf{Exp}^{i,j}_{\mathcal{A}_{\mathsf{MKSE}},\mathsf{Sim}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t)] \\ -\Pr[\mathsf{Exp}^{i,j+1}_{\mathcal{A}_{\mathsf{MKSE}},\mathsf{Sim}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t) = 1] \end{matrix} \right| > \frac{1}{p(\lambda)},$$

for some polynomial function $p(\lambda)$ or $j = m$ and

$$\left| \begin{matrix} \Pr[\mathsf{Exp}^{i,m}_{\mathcal{A}_{\mathsf{MKSE}},\mathsf{Sim}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t)] \\ -\Pr[\mathsf{Exp}^{i+1,1}_{\mathcal{A}_{\mathsf{MKSE}},\mathsf{Sim}_{\mathsf{MKSE}},\mathsf{MKSE}}(1^\lambda, q, t) = 1] \end{matrix} \right| > \frac{1}{p(\lambda)}.$$

Fix one such value of i, j. Consider the relevant owner and client interaction that the simulator is attempting to simulate denoted as $\mathtt{Ow}^*$ and $\mathtt{C}^*$. For the case when $\mathtt{Ow}^*_{\mathcal{A}} = 0$ and $\mathtt{C}^*_{\mathcal{A}} = 0$, the joint view of AcCtrl operations is simulatable as per Figure 3 by $\mathsf{Sim}_{\mathsf{AcCtrl}}$, the overall simulator for MKSE runs $\mathsf{PIR.Setup}$ on the first bucket's portion of $\mathsf{Sim}_{\mathsf{AcCtrl}}$'s output and then runs $\mathsf{PIR.Query}$ on random values. Thus, the above value being non-negligible contradicts either AcCtrl security or PIR security.

Both parties adversarial We also note in the case when both the owner and client are adversarial, the leakage function in Figure 7 includes the multimap, all queries, and all policies as input. The simulator, honestly prepares all of the protocol transcripts in this case, providing the relevant information in addition to the real view. This case cannot be non-negligible as the distance is 0.

Adversarial Client Only We proceed in this case by showing that an $\mathcal{A}_{\mathsf{MKSE}}$ for some adversarial client implies an $\mathcal{A}_{\mathsf{AcCtrl}}$. We then also show that the $\mathsf{Sim}_{\mathsf{AcCtrl}}$ (for this $\mathcal{A}_{\mathsf{AcCtrl}}$) is a good simulator for the MKSE experiment as well. Consider the following construction of $\mathcal{A}_{\mathsf{AcCtrl}}$ for some fixed $\mathtt{Ow}^* \notin \mathtt{Ow}_{\mathcal{A}}$ and $\mathtt{C}^* \in \mathtt{C}_{\mathcal{A}}$.

1. Initialize $\mathcal{A}_{\mathsf{MKSE}}$, receive $\mathtt{Map}$.
2. Sample $\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}}$.
3. For each $\mathsf{w} \in \mathtt{Map}$, run $\tilde{\mathsf{w}} = F(\mathsf{KOwn}_{\mathsf{PRF},\mathsf{DIP}}, \mathsf{w})$.
4. Define $\mathtt{Map}'[\tilde{\mathsf{w}}] = \mathtt{Map}[\mathsf{w}]$ for all $\mathsf{w} \in \mathtt{Map}$.
5. Run $\mu, \sigma, K \leftarrow \mathsf{Size}(|\mathtt{Map}'|)$,
6. $(\mathsf{Buckets}, \mathsf{Stash}) \leftarrow \mathsf{Build}(\mathtt{Map}', K)$.
7. Define π_{DIP} as the implicit permutation defined by Build.
8. For all other pairs of parties where either the index of client or owner is less that $\mathtt{C}^*, \mathtt{Ow}^*$, $\mathcal{A}_{\mathsf{AcCtrl}}$ uses the simulator to prepare the view acting as the challenger in the MKSE experiment. For other interactions, the view is honestly prepared.
9. Send $\mathtt{Map}'$ to experiment. Upon receiving $\mathsf{Docs}_{\mathtt{Out}}$ from challenger. Let $\mathsf{Docs}_{\mathtt{Out},\mathsf{Buckets}}$ denote the portion corresponding to positions in $\mathsf{Buckets}$. Define $\mathsf{Docs}_{\mathtt{Out},\mathsf{Stash}}$ analogously.
10. Run $\mathsf{PIR.Setup}$ on $\mathsf{Docs}_{\mathtt{Out},\mathsf{Buckets}}$, forward response to $\mathcal{A}_{\mathsf{AcCtrl}}$ along with $\mathsf{Docs}_{\mathtt{Out},\mathsf{Stash}}$.
11. For each received w define $\mathsf{Pos}[\mathsf{w}] = \mathsf{DIP.Lookup}(\tilde{\mathsf{w}})$.
12. For $1 \leq i \leq q^{\mathtt{Ow}^*,\mathtt{C}^*}$ receive $\mathsf{w}_i, \mathtt{Plcy}_i, \mathtt{Shr}_i$ forward $\mathtt{Plcy}_i, \mathsf{Pos}[\mathsf{w}_i], \mathtt{Shr}_i$ to experiment.
13. For queries $\mathsf{w}_i^{\mathtt{Ow},\mathtt{C}}$, honestly execute $\mathsf{PIR.Query}$ on the relevant queries, provide the client view of $\mathsf{PIR.Query}$.
14. Provide the client view of Grant operations to $\mathcal{A}_{\mathsf{MKSE}}$.

By the security of AcCtrl, this $\mathcal{A}_{\mathsf{AcCtrl}}$ is simulatable by a simulator $\mathsf{Sim}_{\mathsf{AcCtrl}}$ with the provided leakage. Furthermore, $\mathsf{Sim}_{\mathsf{AcCtrl}}$ represents a valid $\mathsf{Sim}^{\mathtt{Ow}}_{\mathsf{MKSE}}$ for this particular $\mathtt{C} \in \mathtt{C}_{\mathcal{A}}$. Thus, this term being non-negligible contradicts the security of access control.

Adversarial Owner Only Our argument that $\mathcal{A}_{\mathsf{MKSE}}$ implies an $\mathcal{A}_{\mathsf{AcCtrl}}$ follows from the previous case. In the case that $\mathtt{Ow} \in \mathtt{Ow}_{\mathcal{A}}$, let $\mathsf{Sim}_{\mathsf{MKSE}}$ honestly prepares $\mathtt{Map}$ using Setup. Fix some client $\mathtt{C}$. There are two parts of the view that Sim must prepare in Search:

1. The set of queries to the underlying PIR,
2. The calls to the $\mathsf{AcCtrl.Grant}$.

For the underlying PIR, Sim queries random array positions of size s. We discuss the preparation of $\mathsf{AcCtrl.Grant}$ views after discussing PIR.

Then, define $q^{\mathtt{All}} = \sum_{\mathtt{Ow} \in \mathtt{Ow}_{\mathcal{A}}, \mathtt{C} \notin \mathtt{C}_{\mathcal{A}}} q^{\mathtt{Ow},\mathtt{C}}$. Then, this joint set of views is indistinguishable from the true search by defining $\mathcal{A}_{\mathsf{PIR}}$ that provides two query sets either the set of true keywords (translated into locations using K_{DIP}) or random locations. By Definition 3, for all adversaries $\mathcal{A}_{\mathsf{PIR}}$ output 1 with negligibly different probability for $q = s \cdot q^{\mathtt{All}}$. The calls to $\mathsf{AcCtrl.Grant}$ are simulated for each $\mathtt{C} \notin \mathtt{C}_{\mathcal{A}}$ by calling $\mathsf{Sim}^{\mathsf{AcCtrl}}$. Since this $\mathsf{Sim}^{\mathsf{AcCtrl}}$ also represents a good simulator for this hybrid, this term being non-negigible contradicts the security of access control. □

3.2 Preferred Instantiation of MKSE

In the last two subsections, we formally introduced KPIR as our protected map. Our preferred instantiation of MKSE for a MM is to:

1. Use KPIR to protect VolMap.
2. Use KPIR to protect ValMap but use tabulation hashing so the C only sends 2 PIR queries to ValMap and only the response depends on the number of matches.

3. Lastly, we use AcCtrl mechanisms presented in Section 4 to instantiate KPIR.Share.

For w and Vol, the MKSE.Search calls KPIR.Search(w, 1), ..., KPIR.Search(w, Vol) (abusing notation and only showing the client input). The client uses the hashes, H_1, H_2 sampled in DIP.Size to find two positions for each $1 \leq i \leq \texttt{Vol}$, this set of $2 \cdot \texttt{Vol}$ positions is

$$\text{PIR Locations} = \left\{ \begin{matrix} H_1(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, (\mathsf{w}, 1))), ..., H_1(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, (\mathsf{w}, \texttt{Vol}))), \\ H_2(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, (\mathsf{w}, 1))), ..., H_2(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, (\mathsf{w}, \texttt{Vol}))) \end{matrix} \right\}.$$

The client inputs these positions into PIR.Query (and requests the entire stash).

Review of Tabulation Hashing For an input (x_L, x_R), tabulation hashing [PT12, Tho17] hashes each component separately and adds the result: $H(x_L, x_R) = H_L(x_L) + H_R(x_L)$ where the above addition is modulo the number of possible positions. Unfortunately, in the worst case tabulation hashing can increase the stash size in Cuckoo hashing and the resulting DIP [Tho17]. In Section 5, we find this does not happen in our experiments, stash size is very small and does not vary from random hashing.

Tabulation Hashing with Shift Friendly PIR Suppose PIR is *shift friendly* and we replace the set of PIR positions with the following:

$$\text{PIR Locations} = \left\{ \begin{matrix} H_{1,L}(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \mathsf{w})) + H_{1,R}(i), \\ H_{2,L}(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \mathsf{w})) + H_{2,R}(i) \end{matrix} \right\}_{i=1}^{\texttt{Vol}}.$$

Since the S learns the Vol as a result of MUSSE.Search, the client can send queries created as:

$$(\mathsf{q}_{1,L}, \mathsf{st}) \leftarrow \mathsf{PIR.Query}(H_{1,L}(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \mathsf{w}))),$$
$$(\mathsf{q}_{2,L}, \mathsf{st}) \leftarrow \mathsf{PIR.Query}(H_{2,L}(F(\mathsf{KOwn}_{\mathsf{PRF,DIP}}, \mathsf{w}))).$$

S then creates

$$\text{PIR Locations} = \left\{ \mathsf{Query}_{1,i} \leftarrow \mathsf{Shift}(\mathsf{Query}_{j,L}, H_{j,R}(i)) \right\}_{i=1}^{\texttt{Vol}}.$$

This whole set of created positions along with the Vol is returned to C. In the above, public hash functions are crucial.

Resulting Leakage The combined system leaks the following:

1. The size of both maps,
2. The volume of each query,
3. To an adversarial owner, the entire Shr vector.
4. To an adversarial client, all portions of the two maps that they have keys to decrypt. However since the input maps are indexed by a PRF value, the adversary gains no information about possible positions of unqueried keywords.

Table 2 compares this leakage with prior work.

3.2.1 Index Permutation in DEPIR

Definition 5. *Let $\boldsymbol{v} = (v_0, v_1, \cdots, v_{n-1}) \in \mathbb{R}^\psi$ be a vector over a ring $\mathbb{R}$. The circular right shift (also called a cyclic right rotation) of $\boldsymbol{v}$ by k positions, denoted by $Rot_k(\boldsymbol{v})$, is defined as the vector $\boldsymbol{v'} = (v'_0, v'_1, \cdots, c'_{\psi-1})$, where:*

$$v'_j = v_{(j-k) \mod \psi}, \text{ for all } j \in \{0, 1, \cdots, \psi - 1\}$$

When vectors are represented as polynomials in the ring $\mathbb{R}[x]/(x^\psi - 1)$, this operation corresponds to multiplying the polynomial by $x^{-k} \mod (x^\psi - 1)$. That is, if:

$$\boldsymbol{v}(x) = \sum_{i=0}^{\psi-1} v_i x^i,$$

then the circular right shift of $\boldsymbol{v}$ *by* k *positions yields:*

$$\boldsymbol{v}'(x) = x^{-k} \cdot \boldsymbol{v}(x) \mod (x^{\psi} - 1).$$

In the DEPIR protocol, circular right shift plays a central role in making the protocol shift friendly. Specifically, during the query generation phase:

1. The client takes the desired query index $i \in [N]$ written in base-d representation:

$$i = (i_1, \cdots, i_m) \text{ for } m = log_d N$$

2. Each digit i is now encrypted into individual ciphertexts: $\{ct_1, \cdots, ct_m \in R\}$. For all $i = (i_1, \cdots, i_m) \in [N]$, there is a unique polynomial $f_{DB}(X_1, \cdots, X_m)$ such that: $f_{DB}(i_1, \cdots, i_m) = DB[i]$. The resulting ciphertexts encode the index $i' = Rot_k(i)$, that is, the base-d digits of i' are cyclic permutations of those in i.

3. The transformation enables the PIR protocol to be evaluated over the shifted ciphertexts. The server computes:
$$f_{DB}(ct'_1, \cdots, ct'_m) = Enc(DB[i']),$$
where f_{DB} is a polynomial that evaluates to $DB[i']$ at the shifted index i'.

3.2.2 Index Permutation in FrodoPIR

Similar to DEPIR, circular right shift enables query permutation of the client without revealing the target index in FrodoPIR. Specifically, during the query generation phase:

1. The client selects an index $i \in [\mu]$ that it wishes to query. A query vector $f_i = (0, \cdots, 0, \frac{q}{p}, 0, \cdots, 0)$ is formed such that the only non-zero entry is at position i, which is scaled by $\frac{q}{p}$, as per the scheme parameters.

2. To permute the i^{th} position, the client applies a circular right-shift by k positions to the vector f_i, resulting in $f' = Rot_k(f_i)$, such that, the non-zero entry originally at position i now appears at position $(i + k) \mod N$.

3. The shifted vector f' is then used in the protocol in place of f_i, ensuring that the target index is not revealed.

3.2.3 Simultaneously querying multiple Ows

Until now, we have focused on clients interacting with a single Ow. We describe a modification of FrodoPIR to support simultaneous queries to multiple database owners. This enables a single client to query multiple owners in parallel. Roughly, all owners use the same LWE matrix $\mathbf{A}$ and same hashes to create Translate. The size of the response does depend on the number of owners. Since the S does not know which queries match, we do not see a way to compress the response.

The server Setup and Preprocessing steps are as per the original protocol. The only exception is that $\mathbf{A} \in \mathbb{Z}_q^{\psi \times \mu}$ generated by a PRG, is jointly shared by owners. In more detail:

1. For each owner $\texttt{Ow} \in \texttt{Owners}$, an encoded database $\mathbf{D}^{\texttt{Ow}} \in \mathbb{Z}_p^{\mu \times \rho}$ and a vector $\mathbf{M}^{\texttt{Ow}} \leftarrow \mathbf{A} \cdot \mathbf{D}^{\texttt{Ow}} \in \mathbb{Z}_q^{\psi \times \rho}$ are created. We use $\mathbf{M}$ to refer to the concatenation of such $\mathbf{M}^{\texttt{Ow}} \in \mathbb{Z}_q^{n \times \psi \times \rho}$.

2. In the preprocessing phase, the client downloads all $\mathbf{M} \in \mathbb{Z}_q^{n \times \psi \times \rho}$. For each query, the client sample a vector $\mathbf{s} \leftarrow (\chi)^{\psi}$ and $\mathbf{e} \leftarrow (\chi)^{\mu}$. It computes $\mathbf{b} \in \mathbb{Z}_q^{\mu}$ and a matrix $\mathbf{c} \in \mathbb{Z}_q^{n \times \rho}$, where

$$\mathbf{b} \leftarrow \mathbf{s}^T \cdot \mathbf{A} + \mathbf{e}^T,$$
$$\mathbf{c}^{\texttt{Ow}} \leftarrow \mathbf{s}^T \cdot \mathbf{M}^{\texttt{Ow}} \in \mathbb{Z}_q^{\rho}.$$

3. In the query phase, for each intended query index $j \in [\mu]$, the client generates a query vector $\mathbf{f} \in \mathbb{Z}_q^\mu$, where $\mathbf{f} = (0, \cdots, 0, q/p, 0 \cdots, 0)$. Next, it computes the final query as

$$\tilde{\mathbf{b}} \leftarrow \mathbf{b} + \mathbf{f} \in \mathbb{Z}_q^\mu,$$

and sends $\tilde{\mathbf{b}}$ to server.

4. As a response, the server computes a matrix $\tilde{\mathbf{c}}$, where each row $\tilde{\mathbf{c}}^{\mathtt{Ow}}$ is the dot product of $\tilde{\mathbf{b}} \cdot \mathbf{D}^{\mathtt{Ow}}$.

5. In post-processing, the client receives matrix $\tilde{\mathbf{c}}$, and outputs $\mathbf{x} \leftarrow \lfloor \tilde{\mathbf{c}} - \mathbf{c}^{\mathtt{Ow}} \rceil \in \mathbb{Z}_p^{n \times \rho}$.

The client can store $\mathbf{s}$ and receive $\mathbf{M}$ as part of the response from the query and use this to compute $\mathbf{c}^{\mathtt{Ow}}$. We believe the tradeoff between storing $\mathbf{M}$ between queries and sending it is application dependent.

4 Instantiating Access Control Mechanism

In this section, we present two instantiations of AcCtrl mechanism. The first construction (Construction 3) is based on symmetric key cryptography, the second construction uses oblivious PRFs [CHL22, NR04, FIPR05] evaluation between the owner and client. The second construction (Construction 4) allows the owner to control how many keywords keys are provided to each client but does not show the owner the queries. The second mechanism controls the number of decrypted keywords.

4.1 Keyword AcCtrl from PRFs

As mentioned in the definition of AcCtrl, it can be thought of as a generalization of real-or-random security and can be constructed using PRFs. Construction 3 formalizes the scheme. During Setup, the owner generates a PRF key K_{w} for each unique keyword w, and encrypts all the documents that contain w using this key. Later during Grant the owner directly receives a set of queried keywords $\vec{\mathsf{w}}$; for each w that satisfies the $\mathtt{Plcy}$, owner regenerates the same PRF keys and sends them to the client. Client can then decrypt and recover the queried documents.

Construction 3. *Let* $F_{\mathsf{w}} : \{0,1\}^\lambda \times \mathcal{W} \to \{0,1\}^{\eta+\lambda}$ *be a PRF. Let* $\mathsf{Arr}_{\mathtt{Map}} \in \{0,1\}^{\ell \times \eta}$. *Define* $(\mathsf{Setup}, \mathsf{Grant}, \mathsf{Decrypt})$ *as in Figure 8.*

Theorem 3. *Let all variables be as in Construction 3, and consider the following leakage functions:*

$$\begin{aligned}
\mathcal{L}^{\mathsf{AcCtrl},\mathsf{Setup}}(\mathsf{ShrSetup}) &:= \ell, \eta \\
\mathcal{L}^{\mathsf{AcCtrl},\mathsf{Grant}}_{\mathtt{Ow}}(\mathsf{ShrGrant}_i) &:= \mathsf{w}_i \\
\mathcal{L}^{\mathsf{AcCtrl},\mathsf{Grant}}_{\mathsf{C}}(\mathsf{ShrGrant}) &:= \bigcup_{\substack{\mathsf{w} \in \vec{\mathsf{w}}, \\ \mathtt{Plcy}(\mathsf{w}) = 1, \\ i \in \mathsf{Arr}[\mathsf{w}]}} (\mathsf{w}, \pi(i), \mathsf{Arr}[i])
\end{aligned}$$

Then AcCtrl *is a good access control mechanism.*

Proof of Theorem 3. We separately consider correctness and security.

Correctness. For keywords $\mathsf{w}_1, \mathsf{w}_2$ where c is produced using w_1, if $c \oplus K_{\mathsf{w}_2}$ ends with λ 0s it means the first λ bits of $F(\mathsf{KOwn}, \mathsf{w}_1)$ are equal to the first λ bits of $F(\mathsf{KOwn}, \mathsf{w}_2)$. This occurring with noticeable probability contradict the pseudorandomness of F_{w}. Correctness follows by union bound over all such keyword pairs.

Security We separately consider the case where $1_{\mathtt{Ow},\mathcal{A}} = 1$ and $1_{\mathsf{C},\mathcal{A}} = 1$. If neither is 0, then Sim samples a collection of $\ell \cdot (\eta + \lambda)$ random bits as $\mathtt{Map}_{\mathsf{EDocs}}$. This array is indistinguishable from the real output by replacing all K_{w} with random values.

$\underline{\mathsf{Setup}(\mathsf{Arr}_{\mathtt{Map}})}$

1. Sample $\mathsf{KOwn} \leftarrow \{0,1\}^\lambda$.
2. Initialize $\mathsf{CArr}_{\mathsf{EDocs}}$ to be of length ℓ.
3. $\forall 1 \leq i \leq \ell$:
   (a) Let $\mathsf{w} = \mathsf{Arr}_{\mathsf{EDocs}}[i][\mathsf{w}]$
   (b) $\mathsf{CArr}_{\mathsf{EDocs}}[i] = (\mathsf{w}, F_{\mathsf{w}}(K^{\mathtt{Ow}}, \mathsf{w}) \oplus \mathsf{Arr}_{\mathtt{Map}}[i][\mathsf{Doc}]||0^\lambda)$.
4. Output $(\mathsf{CArr}_{\mathsf{EDocs}}, \mathsf{KOwn})$.

$\underline{\mathsf{Grant}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}, \mathtt{Plcy}\\ \vec{\mathtt{Shr}}\end{pmatrix}:}$

$\mathtt{C}$ sends $\vec{\mathtt{Shr}}$ to $\mathtt{Ow}$.

$\mathtt{Ow}$ for each $\mathsf{w} \in \vec{\mathtt{Shr}}$ if $\mathsf{w} \in \mathtt{Plcy}$, set $K_{\mathsf{w}} = F_{\mathsf{w}}(\mathsf{KOwn}, \mathsf{w})$, send K_{w} to $\mathtt{C}$.

$\underline{\mathsf{Decrypt}^{\mathtt{C}}(\vec{c}, \vec{K_{\mathsf{w}}})}$

1. Set $\mathsf{Res} = \perp$
2. For each c in $\vec{c}$
   (a) Parse c as c.
   (b) Let $m = K \oplus c$ for each $K \in \vec{K_{\mathsf{w}}}$.
   (c) If m ends with λ 0s, set $\mathsf{Res} = \mathsf{Res}||m_{1..\eta}$.

Figure 8: Access control from PRFs.

Adversarial Owner Recall $\mathcal{L}^{\mathsf{AcCtrl},\mathsf{Grant}}_{\mathtt{Ow}}(\mathsf{ShrGrant}) = \vec{\mathsf{w}}$. The simulator honestly runs both Setup and Grant rearranging the output of $\mathsf{Arr}_{\mathsf{EDocs}}$ using π and then honestly executes the owner's view of Grant as it sees $\vec{\mathsf{w}}$.

Adversarial Client We assume that each w is only released to a client a single time, the below can be easily modified to be consistent with previous responses. Consider the following simulator:

1. Initialize $\mathsf{CArr}^{\mathsf{No\ w}}$ of size ℓ with random values each of length η.
2. Upon receiving a triple $(\mathsf{w}, i, \mathsf{Doc})$ set $K_w = \mathsf{CArr}[i] \oplus (\mathsf{Doc}||0^\lambda)$ send K_{w} as $\mathtt{View}_j$.

The distribution of ciphertexts and keys prepared by the simulator is the same as the distribution of ciphertexts and keys when the PRF is replaced by a random function. Thus, it holds that,

$$\left|\begin{matrix}\Pr[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{AcCtrl}}(1^\lambda, t, \ell) = 1]\\ -\Pr[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A},\mathsf{Sim},\mathsf{AcCtrl}}(1^\lambda, t, \ell) = 1]\end{matrix}\right| \leq \mathtt{ngl}(\lambda).$$

This completes the proof of Theorem 3. □

4.1.1 Replacing PRF with OPRF

Of course, one can replace the above PRF calls with an OPRF to hide the requested w from the $\mathtt{Ow}$, this same replacement needs to be made on the mapping from w to $\tilde{\mathsf{w}}$ which are used to index the DIP.

Definition 6 (Oblivious Pseudo Random Function (OPRF) [NR04, FIPR05])**.** *Let* OPRF *be a two-party protocol, such that,*

$$\begin{pmatrix}\perp\\ F_{\mathsf{KOwn}}(x)\end{pmatrix} \leftarrow \mathsf{OPRF}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}\\ x\end{pmatrix}.$$

$\underline{\mathsf{Setup}(\mathsf{Arr}_{\mathtt{Map}})}$

1. Sample $\mathsf{KOwn} \leftarrow \{0,1\}^\lambda$.
2. Initialize $\mathsf{Arr}_{\mathsf{EDocs}}$ to be of length ℓ. For all i set
$$\mathsf{Arr}_{\mathsf{EDocs}}[i].\mathsf{w} = \mathsf{Arr}_{\mathtt{Map}}[i].\mathsf{w}.$$
3. For all $\mathsf{w} \in \mathsf{Arr}_{\mathtt{Map}}$, define $K_\mathsf{w} = F_\mathsf{w}(K^{\mathtt{Ow}}, \mathsf{w})$.
4. $\forall 1 \leq i \leq \ell$:
   (a) Sample $r_i \leftarrow \{0,1\}^\lambda$.
   (b) $\mathsf{Arr}_{\mathsf{EDocs}}[i].c = \big(K_w \oplus (\mathsf{Arr}_{\mathtt{Map}}[i].\mathsf{Doc}||0^\lambda)\big)$.
5. Output $(\mathsf{Arr}_{\mathsf{EDocs}}, \mathsf{KOwn})$.

$\underline{\mathsf{Grant}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}, \mathtt{Plcy}\\ \vec{\mathtt{Shr}}\end{pmatrix}:}$

$\mathtt{C}$ for each $\mathsf{w} \in \vec{\mathtt{Shr}}$:

1. $\begin{pmatrix}\perp\\ K_\mathsf{w}\end{pmatrix} \leftarrow \mathsf{OPRF}^{\mathtt{Ow},\mathtt{C}}\begin{pmatrix}\mathsf{KOwn}\\ \mathsf{w}\end{pmatrix}$.
2. If $\mathtt{Plcy} = 0$, $\mathtt{Ow}$ sends $\perp$,
3. Otherwise $\mathtt{Plcy} = \mathtt{Plcy} - 1$ and perform OPRF.
4. $\mathtt{C}$ checks If $K_\mathsf{w} = \perp$, go to next step.

$\mathtt{C}$ does for each c:

1. Set $\mathsf{Res} = \perp$
2. For each c in $\vec{c}$
   (a) Let $m = F_{\mathsf{Docs}}(K, r) \oplus c$.
   (b) If m ends with λ 0s, set $\mathsf{Res} = \mathsf{Res}||m_{1..\eta}$.
3. Output Res.

Figure 9: OPRF construction of access control.

Let F_{KOwn} be non-interactive evaluation with the same outcome. We call OPRF a secure OPRF if

1. *F is a PRF family,*
2. *OPRF outputs $F_{\mathsf{KOwn}}(x)$ with overwhelming probability,*
3. *For every PPT $\mathcal{A}^{\mathtt{Ow}}_{\mathsf{OPRF}}$ that specifies a vector of inputs $\vec{x}, \mathsf{KOwn}$ and observes $\mathtt{Ow}$'s view of OPRF on each input there exists a PPT $\mathsf{Sim}^{\mathtt{Ow}}_{\mathsf{OPRF}}(|\vec{x}|)$ that produces indistinguishable views, and*
4. *For every PPT $\mathcal{A}^{\mathtt{C}}_{\mathsf{OPRF}}$ that specifies a vector of inputs $\vec{x}, \mathsf{KOwn}$ and observes $\mathtt{C}$'s view of OPRF on each input with a uniform random KOwn there PPT $\mathsf{Sim}^{\mathtt{C}}_{\mathsf{OPRF}}(\vec{x}, \vec{F_{\mathsf{KOwn}}}(x))$ that produces indistinguishable views.*

Construction 4. *Let $F_\mathsf{w} : \{0,1\}^\lambda \times \mathcal{W} \to \{0,1\}^\lambda$ and $F_{\mathsf{Docs}} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{\eta+\lambda}$ be oblivious PRFs. Let $\mathsf{Arr}_{\mathtt{MM}} \in \{0,1\}^{\ell \times \eta}$. Define $\mathsf{Setup}, \mathsf{Grant}$ as in Figure 9.*

Theorem 4. *Consider construction 4, OPRF scheme and consider the following leakage functions $\mathcal{L}^{\mathsf{AcCtrl}}_{\mathtt{Ow}}(\mathsf{ShrSetup}) :=$*

$|\vec{w}|$ *and :*

$$\mathcal{L}_{\mathsf{C}}^{\mathsf{AcCtrl},\mathsf{Grant}}(\mathsf{ShrGrant}) := \bigcup_{\substack{\mathsf{w} \in \vec{\mathsf{w}}, \\ \mathtt{Plcy}(\mathsf{w}) = 1, \\ i \in \mathsf{Arr}[\mathsf{w}]}} (\mathsf{w}, \pi(i), \mathsf{Arr}[i])$$

Then $\mathsf{AcCtrl}_{\mathsf{OPRF}}$ *is an access control mechanism.*

Proof of Theorem 4. The proof proceeds analogously to Theorem 3 with additional hybrid steps where one replaces the true transcript of the relevant OPRF with the simulated output. A similar argument is used to show that $\mathcal{A}_{\mathsf{MKSE}}$ corresponds to a valid $\mathcal{A}_{\mathsf{OPRF}}$ (for either the $\mathtt{Ow}$ or $\mathtt{C}$ making $\mathcal{A}_{\mathsf{MKSE}}$ into an $\mathcal{A}_{\mathsf{AcCtrl}}$ and the corresponding $\mathsf{Sim}_{\mathsf{AcCtrl}}$ being valid as an MKSE simulator causing a contradiction.

Adversarial Owner The simulator honestly runs Setup rearranging the output of $\mathsf{Arr}_{\mathsf{EDocs}}$ using π and then uses the $\mathsf{Sim}_{\mathsf{OPRF}}$ to create the client's view of each Grant execution.

Adversarial Client We assume that each w is only released to a client a single time, the below can be easily modified to be consistent with previous responses. Consider the following simulator:

1. Initialize $\mathsf{CArr}^{\mathsf{No\ w}}$ of size ℓ with random values each of length η.
2. Upon receiving a triple $(\mathsf{w}, i, \mathsf{Doc})$, run $\mathsf{Sim}^{\mathsf{C}}_{\mathsf{OPRF}}$ on the leakage and send the response as View_i.

We then consider a further hybrid where the simulator is initialized with $K_w = \mathsf{CArr}[i] \oplus (\mathsf{Doc}||0^{\lambda})$ in place of the OPRF output, this is indistinguishable by security of the OPRF. □

5 Implementation and Experimental Results

We present a proof-of-concept implementation in Rust 1.87.0. We instantiate both `VolMap` and `ValMap` using `KPIR` and use Construction 3 for access control. Instantiating `VolMap` with MUSSE would lower the computation time and yield a single round see Appendix A. The only parallelism is PIR sharding [DPC23].

Throughout, we assume that documents are duplicated to be associated with each instance of w. In our implementation, we encrypt an identifier of the document. Having search revealing identifiers of documents rather than the documents is common in searchable encryption. We do not recommend directly associating documents with each keyword. For computing storage overhead we treat a plaintext map as requiring 40bytes for each keyword document identifier pair, 32 bytes to store a hash of the keyword and 8 bytes for an identifier. Using the Enron dataset, the raw dataset is 2.6GB, associating each keyword with document identifiers is 321MB, if one associates documents with each keyword the size of the map becomes 332GB.

One can add a second round of communication to retrieve the relevant documents. The last round would not require a keyword PIR but would require the owner to authorize decryption of documents (or reuse keys). A secure document retrieval is critical to security of the overall system [GPP21, GPPW24]. SWiSSSE [GPPW24] also requires a third round for document retrieval. For the FNU version of MUSSE [WP21], we consider a tree based oblivious map from Wang et al. [WNL+14] of size 28000 to store `VolMap` with a branching factor of 8, requiring 5 round-trips to `VolMap`. 28000 is the number of different keywords in the Enron dataset.

5.1 Implementation Choices

We utilize Blake3, AES-256, and ChaCha20 as our cryptographic algorithms. We use FrodoPIR [DPC23] as our PIR due to its maturity and flexibility. We did comparative testing with state of the art batch PIR by Mughees and Ren [MR23]. For a database of size 2^{20} a query where $|\mathsf{Docs}_{\mathsf{w}}| = 10000$ Mughees and Ren's implementation takes 85 seconds, which is slightly faster than the same query using FrodoPIR, which took 97 seconds. However, when $|\mathsf{Docs}_{\mathsf{w}}| = 700$ and one scales down the batch size for Mughees and Ren, their

	Setup, Query, Bandwidth, Stash, and Storage Statistics for Enron													
		$\|\mathtt{MM}[\mathsf{w}]\| = 100$			$\|\mathtt{MM}[\mathsf{w}]\| = 500$			$\|\mathtt{MM}[\mathsf{w}]\| = 1000$			Stash			
Size	Setup	Q	BW	BW_{Tab}	Q	BW	BW_{Tab}	Q	BW	BW_{Tab}	Tab.	Rand.	$\|\mathtt{MM}[\mathsf{w}]\|$	Stor.
2^{16}	8	.3	.4	.2	.5	1.4	.3	.9	2.8	.4	81	77	2	0.1
2^{17}	12	.4	.6	.2	.8	2.6	.3	1.5	5.2	.5	73	81	4	0.1
2^{18}	19	.5	1.1	.3	1.4	5.0	.4	2.6	9.9	.6	102	101	8	0.2
2^{19}	30	.8	2.1	.5	2.5	9.8	.6	4.6	19.6	.8	110	123	15	0.2
2^{20}	50	1.2	4.1	.9	4.5	19.4	1.0	8.5	38.8	1.2	111	119	32	0.4
2^{21}	83	2.1	8.1	1.7	8.5	38.6	1.8	15.5	77.2	2.0	152	156	63	0.6
2^{22}	146	3.8	16.1	3.3	16.5	77	3.4	30.5	153.9	3.6	141	160	126	1.2
2^{23}	269	7.2	32.2	6.5	32.8	153.8	6.6	61.5	307.5	6.8	157	165	253	2.2
2^{24}	527	13.9	64.1	12.9	65.1	307.4	13.1	122.9	614.7	13	158	152	506	4.4
24M	778	20.1	93.3	18.8	94.8	447.7	18.9	178	895.3	19	154	154	737	6.3

Table 4: Comparison of Random and Tabulation hashing for Enron Corpus $\epsilon = 0.1$ and $p = 21$. The numbers in column 'Storage' are reported in GB. $|\mathtt{MM}[\mathsf{w}]|$ is the average number matching Docs across w. 24M is the full size of the Enron Corpus. All times are in seconds. Bandwidth (BW) is reported in MB. In Tabulation Bandwidth column, client sends two PIR queries and the server uses tabulation hashes to compute all responses. Queries are over keywords linked to 100, 500, and 1000 documents.

query took 57 seconds while FrodoPIR was 6 seconds. We chose FrodoPIR due to superior performance on representative queries.

We use Cuckoo Hashing [PR01] to build our DIP with support for both random and tabulation hashing. Cuckoo hashing enables the quick setup of our DIP with a small stash size. If Cuckoo hashing produces too large of a stash, the implementation has TethysDIP [BBF+21]. We do note that we are considering smaller parameters than used in TethysDIP's [BBF+21] optimality proof. However, we see very modest stash sizes in all experiments.

All evaluations and benchmarks are performed on a server with AMD Ryzen Threadripper PRO 7995WX CPU with 96 physical cores and 768 GB of RAM, running Ubuntu 22.04. As mentioned above, our benchmarks use only a single thread, except when sharding where we use between $4, 8, 16, 32$ or 128 shards. We utilize the Enron Corpus [KY04] to evaluate our scheme. We do not show results for the synthetic power-law distribution dataset, as they are comparable to the Enron, a real-world example of a power-law distribution. For Enron, we generate subsets with keyword-value maps ranging in size from 2^{16} to 24M, which is $\approx 2^{24.5}$, where the upper bound corresponds to the size of the full Enron Corpus.

To evaluate our scheme, we set a configuration in which each document pointer is encrypted to 32 bytes, along with 16 bytes of initialization vector (IV), with each bucket containing $p = 21$ encrypted pointers, plus an additional 16 bytes of padding, resulting in a total of 1024 bytes of data per bucket. That is, for a multimap of size ℓ DIP creates $(1.8 + \epsilon) \cdot \ell/p$ total positions in $\mathsf{Buckets}$. There are two hash functions in our DIP meaning that each w can be in two $\mathsf{Buckets}$ (or the Stash).

We use the default settings for FrodoPIR, which considers an array where each position is of size 2^{13} bits or 1024 bytes. The resulting DIP buckets are then transformed into a database that can be queried using FrodoPIR. We perform queries for each dataset, considering queries over w where $\mathsf{Docs}_{\mathsf{w}}$ is 100, 500, or 1000. The median keyword for Enron is associated with 21 documents.

5.2 Experimental Results

Table 4 shows our primary results with bandwidth, computation, storage, and stash size. As an example, the first row represents a dataset of size 2^{16}, which takes 8 seconds to setup for a subset of the Enron Corpus. The setup is a one-time process. Following this, it takes .3 seconds for each query on keywords with 100 documents linked to them.

For the full Enron Corpus setup completes in 778 seconds or about 13 minutes. We perform queries to keywords with 100, 500, and 1000 linked documents in $20.1, 94.8$, and 178 seconds, respectively, with a stash of size 154 for both random hashing and tabulation hashing. The Enron Corpus $\ell = 24$ million, so this is

Shards	Setup	Q	Tabulation BW	Tabulation T	Random BW	Random T
$\lvert\texttt{MM}[\mathsf{w}]\rvert = 100$						
1	842	20	18.8	25.7	93	31.6
4	227	5	4.9	6.6	93	13.7
8	124	3	2.7	3.4	93	10.6
16	79	1	1.8	2.3	94	9.7
32	40	0.6	1.6	1.2	94	8.6
64	18	0.3	2.2	0.9	95	8.3
128	11	0.1	3.9	0.8	97	8.2
$\lvert\texttt{MM}[\mathsf{w}]\rvert = 500$						
1	842	95	18.9	115.7	448	150
4	227	24	5.3	29.2	448	64.5
8	124	12	3.5	14.5	449	50.1
16	79	6	3.3	7.5	450	43.2
32	40	3	4.8	3.6	452	39.4
64	18	2	8.6	2.9	456	38.7
128	11	1	17.0	2.6	464	38.3
$\lvert\texttt{MM}[\mathsf{w}]\rvert = 1000$						
1	842	211	19.0	213	895	283
4	227	52	5.8	53	896	124
8	124	26	4.5	27	897	97
16	79	14	5.3	15	899	86
32	40	6	8.8	7	903	78
64	18	4	17.0	6	911	77
128	11	2	33.0	5	927	76

Table 5: Performance metrics across different numbers of shards of Enron Corpus with tabulation hashing. All times are in seconds, bandwidth is in MB. Tabulation columns are when submitting just two PIR queries and using tabulation hashing to generate responses, Rand is the bandwidth for random hashing. Time (T) is computed as $.2 + \text{BW} * 8/100 + \text{Q}$ corresponding to 100ms latency and 100Mb bandwidth. Stash never exceeds 200 in testing.

a stash size of .0006%. Furthermore, we have the storage of 6.3GB and bandwidth of $93,447$, and 895MB for $|\mathsf{Doc_w}| = 100, 500$, and 1000 respectively. We only report bandwidth of the PIR protocol, the stash is of size 7KB and only needs to be sent once. However, by using tabulation hashing the client only needs to send two PIR queries (one for each hash) and the server is able to craft all of the responses. Since for FrodoPIR queries are much larger than responses this yields a drastic savings in BW, to $18.8, 18.9, 19.0$MB for $|\mathsf{Doc_w}| = 100, 500$, and 1000 respectively.

Each query corresponds to two steps: first, retrieving volume from `VolMap`, which involves two PIR queries for the two hash positions; second, retrieving the documents from `ValMap`, which involves $2 \cdot |\mathsf{Docs_w}|/21$ array positions in the PIR. As such, computational time (and bandwidth) scales linearly with the number of matching documents. For example, querying a keyword associated with 100 documents in the entire Enron corpus takes 510 ms using `VolMap`, whereas it takes 19.5 seconds using `ValMap`.

Size of Stash In all our experiments, the stash size never exceeded 200 entries. Similarly, we do not observe a substantive variation in stash size between random and tabulation hashing.

Impact of Tabulation Hashing Somewhat surprisingly, occasionally, tabulation hashing demonstrates a better DIP packing despite theory predicting a higher hash collision probability [Tho17]. We believe this is due to an artifact of converting the cryptographic hashes to work $\mod \mu$ where $\mu = |\mathsf{Buckets}|$. As this number is not a power of 2 it does create non-uniformity in the output. Since tabulation hashing adds two numbers that are both $\mod |\mathsf{Buckets}|$ this can create a slightly more uniform output. To check this across 1000 runs of setup, we find that the distribution of buckets for tabulation hashing has a slightly smaller KL divergence with the uniform distribution than random hashing.

Sharding to Parallelize Search To improve processing time, Table 5 shows performance when using $4, 8, 16, 32, 64$ and 128 PIR shards, one obtains nearly linear speed up with 32 shards, allowing processing of the majority of Enron Corpus queries in $0.6s$ with reasonable bandwidth of < 2MB for keywords associated with 100 documents. Additionally, we answer a query in $1.2s$ for a round trip with $100ms$ of latency, and 100 Mb bandwidth. Such a change also removes the vast majority of setup time (as one operates on smaller matrices). We do not calculate rounds, which is 2 for all our experiments.

Acknowledgments

The authors thank the anonymous reviewers and Ariel Hamlin for their help in improving the manuscript. The work of A.S., M.R., and B.F. has been supported by NSF grants #2141033 and #2232813.

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *IEEE S&P*, pages 962–979. IEEE, 2018.

[AG22] Megumi Ando and Marilyn George. On the cost of suppressing volume for encrypted multi-maps. *PoPETS*, 4:44–65, 2022.

[AGY+22] Nitish Andola, Raghav Gahlot, Vijay Kumar Yadav, S Venkatesan, and Shekhar Verma. Searchable encryption on the cloud: a survey. *The Journal of Supercomputing*, 78(7):9952–9984, 2022.

[APP+23] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *PoPETS*, 1:417–436, 2023.

[BBF+21] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. SSE and SSD: page-efficient searchable symmetric encryption. In *CRYPTO 2021*, pages 157–184. Springer, 2021.

[BHJP14] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):1–51, 2014.

[BIM00a] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[BIM00b] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73. Springer, 2000.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *TCC*, pages 662–693, 2017.

[CD24] Sofía Celi and Alex Davidson. Call me by my name: Simple, practical private information retrieval for keyword queries. In *CCS*, 2024.

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *STOC*, pages 304—-313, 1997.

[CGN97] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. 1997.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS*, pages 668–679, 2015.

[CHL22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. In *EuroS&P*, pages 625–646, 2022.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *CCS*, pages 1223–1237, 2018.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In *TCC*, pages 694–726, Cham, 2017. Springer International Publishing.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, nov 1998.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *CCS*, pages 1243–1255, 2017.

[CWP+21] Javad Ghareh Chamani, Yun Wang, Dimitrios Papadopoulos, Mingyang Zhang, and Rasool Jalili. Multi-user dynamic searchable symmetric encryption with corrupted participants. *IEEE TDSC*, 20(1):114–130, 2021.

[DPC23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, scalable, single-server private information retrieval. *PoPETS*, 2023.

[DPP+16] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. In *ACM SIGMOD/PODS Conference*, 2016.

[DPPS20] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: Attack mitigation for encrypted databases via adjustable leakage. In *USENIX Security*, pages 2433–2450, 2020.

[FIPR05] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324. Springer, 2005.

[FPR+25] Benjamin Fuller, Arinjita Paul, Maryam Rezapour, Ronak Sahu, and Amey Shukla. Mars: Low-leakage multi adversarial owner and reader replication-free searchable encryption from private information retrieval, 2025. `https://github.com/whyamey/mars/`.

[FVK+15] Ben A Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M Bellovin. Malicious-client security in blind seer: a scalable private DBMS. In *IEEE S&P*, pages 395–410. IEEE, 2015.

[FVY+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. SoK: Cryptographically protected database search. In *IEEE S&P*, pages 172–191. IEEE, 2017.

[GKL+20] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *USENIX Security*, pages 2451–2468, 2020.

[GKM21] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In *Eurocrypt*, pages 370–396. Springer, 2021.

[GLMP18] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *CCS*, pages 315–331, 2018.

[GMN+16] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. In *CCS*, pages 1353–1364, 2016.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *STOC*, pages 182–194, 1987.

[GPP21] Zichen Gui, Kenneth G Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. *Cryptology ePrint Archive*, 2021.

[GPP23] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. In *IEEE S&P*, 2023.

[GPPW24] Zichen Gui, Kenneth G Paterson, Sikhar Patranabis, and Bogdan Warinschi. Swissse: System-wide security for searchable symmetric encryption. *PoPETS*, 2024.

[GSB+17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *IEEE S&P*, pages 655–672. IEEE, 2017.

[HSWW18] Ariel Hamlin, Abhi Shelat, Mor Weiss, and Daniel Wichs. Multi-key searchable encryption, revisited. In *PKC*, pages 95–124. Springer, 2018.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *NDSS*, volume 20, page 12. Citeseer, 2012.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *STOC*, pages 262–271, 2004.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *TCC*, pages 668–697. Springer, 2015.

[KE19] Evgenios M Kornaropoulos and Petros Efstathopoulos. The case of adversarial inputs for secure similarity approximation protocols. In *EuroS&P*, pages 247–262. IEEE, 2019.

[KKM+22] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Thomas Schneider, Amos Treiber, and Michael Yonli. Sok: Cryptanalysis of encrypted search with leaker - a framework for leakage attack evaluation on real-world data. In *Euro S&P*, 2022.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *CCS*, pages 1329–1340, 2016.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, pages 818–829, 2016.

[KM18] Seny Kamara and Tarik Moataz. Encrypted multi-maps with computationally-secure leakage. Cryptology ePrint Archive, Paper 2018/978, 2018.

[KMO18] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. Structured encryption and leakage suppression. In *CRYPTO*, pages 339–370. Springer, 2018.

[KMPP22] Evgenios M Kornaropoulos, Nathaniel Moyer, Charalampos Papamanthou, and Alexandros Psomas. Leakage inversion: Towards quantifying privacy in searchable encryption. In *CCS*, pages 1829–1842, 2022.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[KOR+16] Aggelos Kiayias, Ozgur Oksuz, Alexander Russell, Qiang Tang, and Bing Wang. Efficient encrypted keyword search for multi-user data sharing. In *ESORICS*, pages 173–195. Springer, 2016.

[KPT20] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: attacks on encrypted databases beyond the uniform query distribution. In *IEEE S&P*, pages 1223–1240. IEEE, 2020.

[KY04] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *European conference on machine learning*, pages 217–226. Springer, 2004.

[LMM+23] Feng Li, Jianfeng Ma, Yinbin Miao, Ximeng Liu, Jianting Ning, and Robert H Deng. A survey on searchable symmetric encryption. *ACM Computing Surveys*, 56(5):1–42, 2023.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. In *STOC*, page 595–608, 2023.

[LXY+23] Feng Liu, Kaiping Xue, Jinjiang Yang, Jing Zhang, Zixuan Huang, Jian Li, and David SL Wei. Volume-hiding range searchable symmetric encryption for large-scale datasets. *IEEE TDSC*, 2023.

[MK22] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In *USENIX Security 22*, pages 1723–1740, 2022.

[MR23] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *IEEE S&P*, pages 437–452. IEEE, 2023.

[MT19] Evangelia Anna Markatou and Roberto Tamassia. Full database reconstruction with access and search pattern leakage. In *ISC*, pages 25–43. Springer, 2019.

[MVA+23] Sujaya Maiyya, Sharath Chandra Vemula, Divyakant Agrawal, Amr El Abbadi, and Florian Kerschbaum. Waffle: An online oblivious datastore for protecting data access patterns. *Proceedings of the ACM on Management of Data*, 1(4):1–25, 2023.

[MZRA22] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. Incremental offline/online PIR. In *USENIX Security*, pages 1741–1758, 2022.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, March 2004.

[OPPW24] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. In *Financial Crypto*, 2024.

[PPY18] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Symmetric searchable encryption with sharing and unsharing. In *Euro S&P*, pages 207–227. Springer, 2018.

[PPYY19] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *CCS '19*, 2019.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *European Symposium on Algorithms*, pages 121–133. Springer, 2001.

[PSV+14] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *NSDI*, pages 157–172, 2014.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *Eurocrypt*, pages 125–157. Springer, 2018.

[PSY23] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be dense: Efficient keyword PIR for sparse databases. In *USENIX Security 23*, pages 3853–3870, 2023.

[PT12] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM (JACM)*, 59(3):1–50, 2012.

[PZ13] Raluca Ada Popa and Nickolai Zeldovich. Multi-key searchable encryption. *Cryptology ePrint Archive*, 2013.

[SWP00] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P*, pages 44–55. IEEE, 2000.

[Tho17] Mikkel Thorup. Fast and powerful hashing using tabulation. *Communications of the ACM*, 60(7):94–101, 2017.

[UCK+21] Erkam Uzun, Simon P Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee. Fuzzy labeled private set intersection with applications to private real-time biometric search. In *USENIX Security*, pages 911–928, 2021.

[VRMÖ17] Cédric Van Rompay, Refik Molva, and Melek Önen. A leakage-abuse attack against multi-user searchable encryption. *PoPETS*, 2017.

[WLD+17] Guofeng Wang, Chuanyi Liu, Yingfei Dong, Hezhong Pan, Peiyi Han, and Binxing Fang. Query recovery attacks on searchable encryption based on partial knowledge. In *SecureComm*, pages 530–549. Springer, 2017.

[WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226, 2014.

[WP21] Yun Wang and Dimitrios Papadopoulos. Multi-user collusion-resistant searchable encryption with optimal search time. In *AsiaCCS*, pages 252–264, 2021.

[WP23] Yun Wang and Dimitrios Papadopoulos. Multi-user collusion-resistant searchable encryption for cloud storage. *IEEE Transactions on Cloud Computing*, 2023.

[Yeo23] Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In *CRYPTO*, pages 197–230. Springer, 2023.

A Single-Round MKSE: Modified MUSSE [WP21]

In this section, we show that we can build an MKSE scheme from MUSSE [WP21] and PIR that reduces the round of interaction to one round.

We now review the state of the art MKSE due to Wang and Papadopoulos [WP21]. We modify their scheme to provide security when the Ow is adversarial. This is a simple change that does not impact storage or search performance.

A.1 Modified MUSSE [WP21]

We present a modified version of MUSSE with *No Forward secrecy and No User storage* in Figure 10. Our modifications include: 1) Sampling client keys by clients and passing them from client C to owner Ow whenever a Share is needed. 2) Using maps instead of multimaps and removing the dependency on frequency of keywords in shared documents for data retrieval. And 3) replacing a response hiding map with a response revealing one during search, to reduce the interaction to one round.

Roughly, the leakage is as follows:

1. The size of the map.
2. When a keyword is searched, the server learns the timestamp of the last Shr that involved that keyword and sees the accessed location.
3. For each shared keyword, the server learns the set of timestamps this keyword was shared to the client.
4. An adversarial owner learns $\vec{\texttt{Shr}}$.

$$\frac{\left(\mathsf{K}^{\mathtt{Ow}}\right) \leftarrow \mathsf{KeyGen}^{\mathtt{Ow}}\left(1^{\lambda}\right)}{\left(\mathsf{K}^{\mathtt{C}}\right) \leftarrow \mathsf{KeyGen}^{\mathtt{C}}\left(1^{\lambda}\right)}$$
$$\frac{\begin{pmatrix} \perp \\ \mathtt{EMM} \end{pmatrix} \leftarrow \mathsf{Setup}^{\mathtt{Ow},\mathtt{S}}\begin{pmatrix} \mathsf{M} \\ \perp \end{pmatrix}}{}$$

1. For each $\mathtt{C}_i$: $\mathsf{Access}_{\mathtt{C}_i} \leftarrow []$.
2. $\mathsf{Access} \leftarrow \{\mathsf{Access}_{\mathtt{C}}\}_{\mathtt{C} \in \mathtt{Clients}}$.
3. $\mathtt{EMM} = (\mathsf{DictW}, \mathsf{Access})$

$$\frac{\begin{pmatrix} \perp \\ \mathtt{EMap} \\ \mathsf{K}^{\mathtt{C}} \end{pmatrix} \leftarrow \mathsf{Share}^{\mathtt{Ow},\mathtt{S},\mathtt{C}}\begin{pmatrix} \mathsf{K}^{\mathtt{Ow}}, \mathtt{Plcy} = (\mathtt{C}, D) \\ \mathtt{EMap} \\ \mathsf{K}^{\mathtt{C}}, \mathsf{w} \end{pmatrix}}{}$$

Ow does:

1. $\mathsf{Access}_{\mathtt{C}} \leftarrow \mathsf{Access}_{\mathtt{C}} \cup \{i\}$
2. Send $\mathsf{KW}_{\mathsf{Docs}}(i)$ to client $\mathtt{C}$.

C does:

1. $\mathsf{KeyValue} \leftarrow \perp$
2. For $\mathsf{w} \in \mathsf{KW}_{\mathsf{Docs}}(i)$:
   (a) $\mathsf{addr} \leftarrow G_{\mathsf{K}^{\mathtt{C}}}(0||\mathsf{w})$.
   (b) $\mathsf{val} \leftarrow i \oplus G_{\mathsf{K}^{\mathtt{C}}}(1||\mathsf{w})$.
   (c) $\mathsf{KeyValue} \leftarrow \mathsf{KeyValue} \cup (\mathsf{addr}, \mathsf{val})$.
3. Send $\mathsf{KeyValue}$ and $\mathtt{C}$ to the server.

S does:

1. Insert all $(\mathsf{addr}, \mathsf{val}) \in \mathsf{KeyValue}$ into to DictW.
2. $\mathsf{Access}_{\mathtt{C}} \leftarrow \mathsf{Access}_{\mathtt{C}} \cup \{i\}$.

$$\frac{\begin{pmatrix} \perp \\ \mathsf{CRes} \end{pmatrix} \leftarrow \mathsf{Search}^{\mathtt{S},\mathtt{C}}\begin{pmatrix} \mathtt{EMap} \\ \mathsf{K}^{\mathtt{C}}, \mathsf{w} \end{pmatrix}}{}$$

C does:

1. Send $G_{\mathsf{K}^{\mathtt{C}}}(0||\mathsf{w})$ to $\mathtt{S}$.

S does:

1. $\mathsf{Res}_{\mathsf{w}} = \{\}$, $i = 0$.
2. For each received value addr:
   (a) If $\mathsf{DictW}[\mathsf{addr}] = \mathsf{NULL}$:
   Send "stop" to $\mathtt{C}$ and break.
   (b) Else: $i = \mathsf{DictW}[\mathsf{addr}] \oplus G_{\mathsf{K}^{\mathtt{C}}}(0||\mathsf{w})$.
   $\mathsf{Res}_{\mathsf{w}} = \mathsf{Res}_{\mathsf{w}} \cup i$.
3. Send $\mathsf{Res}_{\mathsf{w}}$ to $\mathtt{C}$.

Figure 10: MUSSE [WP21] protocol.

5. An adversarial client learns the subset of the map that has been shared with them.

The leakage is formally presented in $\mathcal{L}_{\mathsf{MUSSE}}$ in Figure 7.

Theorem 5. *Let* $\mathsf{MUSSE} = (\mathsf{Setup}_{\mathsf{MUSSE}}, \mathsf{Share}_{\mathsf{MUSSE}}, \mathsf{Search}_{\mathsf{MUSSE}})$ *be defined as Figure 10 with leakage function* $\mathcal{L}^{\mathsf{MUSSE}}(\cdot)$. MUSSE *is a* MKSE *scheme.*

Proof. Consider the MUSSE construction in Figure 10, and some adversary $\mathcal{A}_{\mathsf{MUSSE}}$. Our goal is to build a simulator $\mathsf{Sim}_{\mathsf{MKSE}}$ that satisfies the definition 1 such that for all $q, t = \mathtt{poly}(\lambda)$:

$$\left| \begin{array}{l} \Pr[\mathsf{Exp}_{\mathsf{Sec},\mathcal{A}_{\mathsf{MUSSE}},\mathsf{MUSSE}}(1^\lambda, q, t) = 1] \\ -\Pr[\mathsf{Exp}_{\mathcal{A}_{\mathsf{MUSSE}},\mathsf{Sim}_{\mathsf{MUSSE}},\mathcal{L}_{\mathsf{MUSSE}}}(1^\lambda, q, t) = 1] \end{array} \right| \leq \mathtt{ngl}(\lambda).$$

Adversarial Client We prove this by the following hybrids:

1. Let hybrid $\mathcal{H}_0$ be the real execution of the protocol.
2. Assume that hybrid $\mathcal{H}_1$ is the same as hybrid $\mathcal{H}_0$ with the following differences: The challenger keeps a list of $L = (\mathsf{addr}, \mathsf{val}, \mathtt{C}, \mathsf{w}, i, \mathsf{op})$ and updates it for each $\mathsf{op} = \mathsf{Share}/\mathsf{Search}$. The output of the PRF $G^{\mathtt{C}}_{\mathsf{K}}$ in the real game is replaced by a uniformly random value.

   During the initial Share operation, the challenger inserts a new entry for each shared keyword to the list. Later, upon searching w, the simulator runs a linear search over the values assigned to each keyword, if nothing was found it adds a new tuple including keyword w along with random $\mathsf{addr}, \mathsf{val}$, and $i = \mathsf{NULL}$ to the list; otherwise it retrieves the document id i and insert the new entry for $\mathsf{op} = \mathsf{Search}$ accordingly. Finally, it sends the $\mathsf{addr}, \mathsf{val}$ pairs to the adversary.

   Later, upon further Share operations, the simulator scans L and the challenger checks if there exist any tuple in the list that $i = \mathsf{NULL}$ for $\mathsf{w} \in \mathsf{Docs}_i$ with $\mathsf{op} = \mathsf{Search}$ and updates its corresponding i. Also, the challenger adds a new tuple to the list for $\mathsf{op} = \mathsf{Share}$.

   Since, in the real game, $G^{\mathtt{C}}_{\mathsf{K}}$ is only computed once for each w, and because the adversary does not have access to the PRF key K, then hybrid $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_0$.
3. Let hybrid $\mathcal{H}_2$ be the ideal execution of the game as described in the security definition 1 where the stateful simulator only receives the leakage functions described in Figure 7. Then all the operations described in hybrid $\mathcal{H}_1$ can also be executed using the leaked data as input. For each $\mathtt{C}$, the simulator sees the timestamp of the Share and Search operations for each shared and searched keyword, including the location accessed on the map. Thus, hybrid $\mathcal{H}_2$ is indistinguishable from hybrid $\mathcal{H}_1$

Adversarial Owner In the case of $\mathtt{Ow} \in \mathtt{Ow}_{\mathcal{A}}$, the simulator first adds the keywords $\mathsf{w} \in \vec{\mathtt{Shr}}^{\mathtt{Ow},\mathtt{C}}$ obtained from the leakage function to its stored list $(\mathsf{addr}, \mathsf{val}, \mathtt{C}, \mathsf{w}, i, \mathsf{op})$ under a share operation $\mathsf{op} = \mathsf{Share}$. Then the simulator, honestly fills the rest of the map upon each Share and Search based on the leakage functions described in Figure 7. Since in both cases the map is being filled consistent to the MUSSE protocol, the ideal experiment is indistinguishable from the simulated experiment.

□

A.2 Instantiation

In the last subsection, we formally introduced the protected maps: MUSSE and in section 3 we introduced $\mathtt{KPIR}$. In addition, MUSSE suggests two further options for the $\mathtt{VolMap}$: 1) local storage and 2) a fully oblivious map (OMap) based on storing a tree in oblivious RAM [WNL+14]. For the Enron dataset $|\mathtt{VolMap}| \approx 2^{18}$ while $|\mathtt{ValMap}| \approx 2^{25}$ so local storage of just $\mathtt{VolMap}$ maybe feasible. Alternatively, clients can store results after querying to prevent query equality leakage [KMO18]. An instantiation of MKSE for a $\mathtt{MM}$ that reduces the rounds of interaction to one round is as follows:

1. Use MUSSE to protect $\mathtt{VolMap}$. As a reminder, we modify MUSSE in Figure 10 to:
   (a) Change sharing to support adversarial owners,
   (b) Use one round of communication, and
   (c) Be response revealing so the server directly learns the volume.

2. Use KPIR to protect ValMap.

3. Lastly, we use AcCtrl mechanisms presented in Section 4 through Construction 3 or 4, to instantiate KPIR.Share.

We modify MUSSE to collapse the lookups of VolMap, ValMap into a single communication round. For w and Vol, the MKSE.Search calls KPIR.Search(w, 1), ..., KPIR.Search(w, Vol) (abusing notation and only showing the client input). The client uses the hashes, H_1, H_2 sampled in DIP.Size to find two positions for each $1 \leq i \leq \texttt{Vol}$, this set of $2 \cdot \texttt{Vol}$ positions is

$$\text{PIR Locations} = \begin{Bmatrix} H_1(\mathsf{w}, 1), ..., H_1(\mathsf{w}, \texttt{Vol}), \\ H_2(\mathsf{w}, 1), ..., H_2(\mathsf{w}, \texttt{Vol}) \end{Bmatrix}.$$

The client inputs these positions into PIR.Query (and requests the entire stash).