A Practical Model for Collaborative Databases: Securely Mixing, Searching and Computing

Shweta Agrawal[1]*, Rachit Garg[2]**, Nishant Kumar[3]*, and Manoj Prabhakaran[4]

[1] IIT Madras, India, shweta.a@cse.iitm.ac.in
[2] UT Austin, US, rachit0596@gmail.com
[3] Microsoft Research, India, nishant.kr10@gmail.com
[4] IIT Bombay, India, mp@cse.iitb.ac.in

Abstract. We introduce the notion of a *Functionally Encrypted Datastore* which collects data *anonymously* from *multiple data-owners*, stores it encrypted on an untrusted server, and allows untrusted clients to make *select-and-compute* queries on the collected data. Little coordination and no communication is required among the data-owners or the clients. Our notion is general enough to capture many real world scenarios that require controlled computation on encrypted data, such as is required for contact tracing in the wake of a pandemic. Our leakage and performance profile is similar to that of conventional *searchable encryption* systems, while the functionality we offer is significantly richer.

In more detail, the client specifies a query as a pair (Q, f) where Q is a filtering predicate which selects some subset of the dataset and f is a function on some computable values associated with the selected data. We provide efficient protocols for various functionalities of practical relevance. We demonstrate the utility, efficiency and scalability of our protocols via extensive experimentation. In particular, we evaluate the efficiency of our protocols in computations relevant to the *Genome Wide Association Studies* such as Minor Allele Frequency (MAF), Chi-square analysis and Hamming Distance.

1 Introduction

Many real world scenarios call for performing controlled computation on encrypted data belonging to multiple users. A case in point is that of contact tracing to control the COVID-19 pandemic, where cellphone users may periodically upload their (space, time) co-ordinates to enable tracing of infected persons, but desire the assurance that this data will not be used for any other purpose. Another example is Genome Wide Association Studies (GWAS), which look into entire genomes across different individuals to discover associations between genetic variants and particular diseases or traits [10].

More generally, enabling controlled computation on large-scale, multi-user, encrypted cloud storage is of much practical value in various privacy sensitive situations. Over the last several years, several tools have emerged that offer a variety of approaches towards this problem, offering different trade-offs among security, efficiency and generality. While theoretical schemes based on modern cryptographic tools like secure multi-party computation (MPC), [63,25] fully homomorphic encryption (FHE) [24] or functional encryption (FE) [51,8] can provide strong security guarantees, their computational and communication requirements are often prohibitive for large-scale data (see Section 8.2, and Footnote 19). At the other end are efficient tools like CryptDB [50], Monomi [58], Seabed [46] and Arx [49], which add a lightweight encryption layer under the hood of conventional database queries, but offer only limited security guarantees and do not support collaborative databases (see Table 1). While there also exist tools which seek to strike a different balance by trading off some efficiency for more robust security guarantees and better support for collaboration – like Symmetric Searchable Encryption (SSE) [56,20], and Controlled Functional Encryption (CFE) [43] – they offer limited functionality.

Our Approach: We introduce *Functionally Encrypted Datastores* (FED), opening up new possibilities in secure cloud storage. FED is a secure cloud-based data store that can collect data *anonymously* from *multiple*

* Part of work done at IIT Delhi
** Work done primarily at IIT Madras

data-owners, store it encrypted on untrusted servers, and allow untrusted clients to make *select-and-compute* queries on the collected data. Little coordination and no communication is required among the data-owners or the clients. We obtain a performance and leakage profile similar to that of conventional searchable encryption systems, but with additional anonymity guarantees to support multiple data-owners, and significantly richer computation functionality along with searchability.

The "select and compute" functionality we support may be viewed as typical (relational) database operations on a single table. A query is specified as a pair (Q, f) where Q is a *filtering predicate* which selects some rows of the table, and f is a *function on the selected data.* A key feature we seek is that the computation overheads for a select-and-compute query *should not scale with the entire database size, but only with the number of selected records.*

Some Motivating scenarios: There are many scenarios today where data is collected from users in a centralized database and made available to other users for querying.

- Census Data: governments use census data for administrative purposes, and also provide restricted access to select researchers for select purposes [59,66],
- Contact tracing: Records which have a (space, time) pair from a set of high-risk pairs are filtered and then used for notification for quick tracing.
- Customer data: private corporations can and do collect large amounts of information about their customers, which could be legitimately useful in improving their customer service,
- Genetic Data: *Genome Wide Association Studies (GWAS)* look into entire genomes across different individuals to discover associations between genetic variants and particular diseases or traits [10] (This is the example we use in our experiments.)

In all such scenarios, individuals' privacy is vulnerable, and in the absence of cryptographic measures, entirely reliant on the central authority's good intentions and security.

1.1 Our Model

Before we describe our model, we outline our desiderata:

- *No central trusted authority.* The data-owners should not be required to trust any central server with their private data. However, they may rely on servers to facilitate collecting data and answering queries (without learning the data or the queries).
- *No coordination among offline data-owners.* The data-owners should not be required to trust, or even be aware of each other. Additionally, the data owners need not be online when queries arrive.
- *Untrusted clients, oblivious to the data-owners.* The data-owners should not be required to trust or even be aware of the clients who will query the datastore in the future.
- *Efficient on large scale data.* Enable handling of large databases (e.g. genomic data from hundreds of thousands of individuals, or census data) efficiently. This may be done by allowing some well-defined *leakage* to the servers, as is common in the searchable encryption literature. Typically, the leaked information pertains to whether the same piece of data matches multiple search queries (without revealing what the search queries are). We shall settle for similar leakage in our system as well.
- *Anonymity of data.* As we allow multiple data-owners, the data items referenced in the leakage above when obtained by a compromised server should not be traced back to the data-owners who contributed those items.

A Necessary Relaxation. It is *impossible* to satisfy the first three requirements using a single (untrusted) server – it may internally play the role of clients and make any number of arbitrary queries to the database, violating security. Hence, we propose a solution with two servers: one with large storage that stores all the encrypted data, and the other an auxiliary server that stores only some key material. Either server by itself can be corrupt, but they will be assumed not to collude with each other. This model is well-accepted in the literature for privacy-preserving computation on genomic data [32,16], justified by the fact that in the real world, genomic data in the US [32] may be managed by distinct governmental organizations within the National Institutes of Health and the World Health Organization which are expected not to collude. More generally, this model has been used widely in the literature, including multi-server Private Information Retrieval [17], CFE [43],[5] Searchable Encryption [45], Secure Outsourced Computation [36] (including specifically in the context of genomic data [57]), and even large-scale real-world systems like Riposte [19] and Splinter [62].

1.2 Our Results

As discussed above, our first contribution is the notion of a Functionally Encrypted Datastore (FED), which permits a data-owner to securely outsource its data to a storage server, such that, with the help of an auxiliary server, clients can carry out select-and-compute queries efficiently. We emphasize that our database is *anonymously collaborative* in the sense that it contains data belonging to multiple data owners but hides the association of the (encrypted) data items with their owners. In addition to this, our contributions include:

- A *general framework* for instantiating FED schemes. The framework is modular, and consists of two components, which may be of independent interest – namely, Searchably Encrypted Datastore (SED) and Computably Encrypted Datastore (CED). We present several constructions to instantiate this framework (see an overview in Section 1.3).
- We build a system[6] for our proposed framework and demonstrate the utility, efficiency and scalability of our protocols via *extensive experimentation*. In particular, we study *Genome Wide Association Studies* (GWAS), and demonstrate that genomic records from 100,000 data owners (each contributing a single record) can be securely setup in around 500s, while a client query that filters up to 12,000 records can be answered in less than 15s with a maximum of 15 MB of total communication in the system. For standard functionalities in GWAS, like Minor Allele Frequency (MAF) and Chi Square Analysis, our single data owner based FED protocol has an overhead of merely 1.5×, compared to SSE schemes (which offer no compute functionality).

To the best of our knowledge, no prior work achieves the features of collaborative databases, select and compute functionality and efficiency that we achieve in this work. See Table 1 for a comparison.

1.3 Overview of Constructions

We present several modular constructions of FED (and the single data-owner version sFED), which can be instantiated by plugging in different implementations of its components. Below we give a roadmap of how the two simpler primitives, Searchably Encrypted Datastore (SED) and Computably Encrypted Datastore (CED) can be securely dovetailed into a construction of FED.

The starting point of our constructions are single data-owner versions sSED and sCED. We show that these components can be implemented by leveraging constructions from the literature, namely, the Multi-Client SSE (MC-SSE) scheme due to Jarecki et al. [33] and CFE due to Naveed et al. [43]. The search query family supported by our sSED constructions are the same as in [33]. For sCED, we support a few specialized functions, as well as a general function family. The primitives sSED and sCED are of independent interest, and they can also be combined to yield a single data-owner version of FED (called sFED).

To upgrade sSED and sCED constructions into full-fledged (multi data-owner) SED and CED schemes, we require several new ideas. One challenge in this setting is to be able to hide the association of the (encrypted) data items with their data-owners. A simple approach, wherein each data-owner sends encryptions/secret shares of its data directly to the two servers, hence does not work. Our approach is to first securely merge the data from the different data-owners in a way that removes this association, and then use the single data-owner constructions on the merged data set. For this, both SED and CED constructions rely on *onion secret-sharing techniques*. Onion secret-sharing is a non-trivial generalization of the traditional mix-nets [15]. In a mix-net, a set of senders $\mathsf{D}_1, \cdots, \mathsf{D}_m$ want to send their messages $M_1, \cdots, M_m$ to a server S, with the

[5] In CFE, the storage server is implicit as it carries out no computations.

[6] Implementation available at `https://github.com/RachitG54/FED`

help of an auxiliary server A (who does not collude with S), so that neither S nor A learns the association between the messages and the senders. We require the following generalization: each sender D_i wants to *secret-share* its message M_i between two servers S and A; that is, it sets $M_i = \sigma_i \oplus \rho_i$, and wants to send ρ_i to S and σ_i to A. While the senders want their messages to get randomly permuted in order to remove the association of data with themselves, the association between σ_i and ρ_i needs to be retained. As described in Section 6.2, onion secret-sharing provides a solution to this problem (and more).

In the case of CED, merging essentially consists of a multi-set union which can be solved using onion secret-sharing. But in the case of SED, merging entails merging "search indices" on individual data sets into a search index for the combined data set. A search index is a function that maps a keyword to a set of records; merging such indices requires that for each keyword, *the sets corresponding to it in the different indices* be identified and merged together. For this onion secret-sharing alone is not adequate. We propose two approaches to merge the indices – one in the random oracle model using an Oblivious Pseudorandom Function (OPRF) protocol, and another one with comparable efficiency in the standard model, relying on 2-party Secure Function Evaluation (SFE). Please see Section 6 for more details.

2 Related Work

We contrast our notion of FED with prior work in terms of various features. A detailed summary is tabularized in Table 1.

- *Multiple data owners*: We allow encrypted data to be collected anonymously from multiple data-owners. Prior works that allow this include multi-party computation [63], multi-input functional encryption [27], and controlled functional encryption (CFE) [43], all of which suffer costs proportional to the entire dataset. Multi-key/multi-user Searchable Encryption [29,60] is another line of work, which allows data to be shared from multiple data owners to clients for searching of single keywords, but unlike our work, does not hide the association of the data to its owner, nor allows data owners to be oblivious of the clients.
- *Rich functionality with strong security*: Symmetric searchable encryption (SSE) [20,12,47] and its extensions (including Structured Encryption [13]) support keyword searches and have performance sublinear in the size of the dataset, but do not allow computation on the search results. In contrast, tools such as CryptDB [50], Monomi [58] and Seabed [46] do allow full-fledged search-*and*-compute, but incur higher leakage to the server and argue security in the "snapshot attacker model", which has been criticized for being unrealistically weak [44,28]. Moreover, their security analysis assumes fully trusted clients who do not collude with the server(s). In contrast, we offer a stronger security model where clients may be malicious, either server can collude with a subset of data owners and/or clients and the attacker is persistent, i.e. has access to the view of the corrupt parties throughout the lifetime of the system. Leakage to servers is limited to the leakage typical in SSE, and there is no leakage to data owners or clients.
- *Computationally light clients*: Our clients are very efficient and only perform work proportional to the size of their query and the output they receive, typically independent of database size. In comparison, the CryptDB family of constructions [50] require clients to download certain intermediate results, decrypt them and perform the remaining computation. Similarly, the CFE scheme of [43] requires clients to evaluate a garbled circuit on the entire dataset. While multi-client SSE has more lightweight clients than the above, the clients still do work proportional to the size of the filtered data.
- *Deployability*: We allow clients to join the system dynamically – indeed, clients and data owners can be oblivious of each others' identity (or even number) in contrast with prior constructions [29,60], which assume clients to be fixed at the start of the protocol and the data owner to know about all such clients.

GWAS from other primitives: Prior works have considered GWAS using cryptographic techniques like HE [38,9] and MPC [57]. [32,16] study GWAS computation in a (non-colluding) servers model and MPC computation. However, these lines of work are fundamentally different from ours in that they do not allow clients to search on the data before computing on it. This not only avoids the complex search functionality, but also simplifies the computation task significantly, as securely computing statistics on a fixed data-set scales independently of the actual number of data records. As such, we do not compare our efficiency the with them in this paper.

Table 1: Comparison of related works

	Functionality					Security				Query-Phase Efficiency**	
	Search-and-Compute	Search supports Boolean formulas	Compute supports general functions	Multi Data-Owner	Multi Client	Corruption Level			Notes/Additional Assumptions	Client	Server
						Server(s)	Client(s)	Data-Owner(s)			
CryptDB [50]	Y	Y	Y (SQL)	N	N	Passive	Passive	Passive	Trusted application server and passive DBMS server. No server-client collusion. Only snapshot attacker.	$O(1)$	$O(t)$
Seabed [46]	Y	Y	Y (OLAP)	N	Y	Passive	Passive	Passive	No server-client collusion. Clients need a key given by the data owner. Only snapshot attacker.	$O(1)$	$O(n)$
Arx [49]	Y	Y	N	N	N	Passive	Passive	Passive	Trusted application server and passive DBMS server. No server-client collusion. Only snapshot attacker.	$O(1)$	$O(t \log n)$
OXT [12]	N	Y	NA	N	N	Passive	Passive	Passive	-	$O(t_1 k)$	$O(t_1 k)$
OSPIR [33]	N	Y	NA	N	Y	Passive	Malicious	Malicious	No server-client collusion. Also provides query privacy to clients assuming no server-data owner collusion.	$O(t_1 k)$	$O(t_1 k)$
BlindSeer [47,23]	N	Y	NA	N	Y	Passive	Malicious	Passive	No server-client collusion. Also provides query privacy to clients assuming no server-data owner collusion.	$O(t_1 \log n)$	$O(t_1 \log n)$
CFE [43]	N	NA	Y (circuits)	Y	Y	Passive	Malicious	Passive	No authority-client or authority-storage collusion. Can provide query privacy to clients.	$O(n)$	$O(n)$
Multi-key/user SE [29,60]	N	N	N	Y	Y	Passive	Passive	Malicious	File sharing setting. Server allowed to collude with subset of data owners or clients. DOs aware of clients (fixed at start of protocol). Shared data can be traced back to original DO.	$O(1)$	$O(n)$
This work	Y	Y	Y (circuits)	Y	Y	Passive	Malicious	Passive	Two non-colluding servers. Either server can collude with subset of clients/data owners. Can provide search/function query privacy to clients, anonymity to data owners. DOs oblivious of clients (can dynamically join system).	$O(1)$	$O(t_1 k)$

**Based on a query of the form SELECT SUM(COL) WHERE $\text{COL}_1 = \alpha_1$ AND $\text{COL}_2 = \alpha_2 \cdots$ $\text{COL}_k = \alpha_k$, where COL and COL_i are column names. If computation is not supported, SUM is omitted; if search is not supported WHERE clause is omitted. n denotes the total number of records in the database and t denotes the number of records matching the search condition. WLOG, assume $\text{COL}_1 = \alpha_1$ filters the least number of records and t_1 denotes the number of records satisfying it. In each case, assume that all supported pre-processing has been done prior to the query.

3 Preliminaries

A detailed index of the notation used in our paper is provided in Table 2. Below we give an overview of some of the cryptographic primitives that we will use in our constructions.

Secure 2-party computation (2PC) Secure 2PC allows 2 parties to securely compute a joint function of their private inputs, without revealing anything except the output to the other party. One of the ways to achieve this is by using Yao's Garbled Circuits (GCs) [63], which has been extensively studied [63,4,31,41] and has moreover been optimized for performance by a host of recent works [39,5,65].

Secret sharing Secret sharing [52,7] refers to methods of distributing a secret to multiple participants, each of whom is given a *share* of the secret. An individual share reveals nothing, but when a group of *authorized* particpants combine their shares, the secret may be reconstructed. For the purpose of our work, *additive-secret sharing* suffices, whereby a secret value x is split into random shares x_0 and x_1 s.t. $x_0+x_1=x$.

Symmetric Searchable Encryption (SSE) In SSE [56,20,35], a (single) client offloads storage of encrypted data to an untrusted server such that it may later perform search queries on this data. Typically SSE permits some principled leakage, like access and query patterns of the data, to the server. A multi-client version of SSE was proposed by Jarecki et al. [33]: here, the data owner is separate from (and does not trust) the clients and remains online throughout the protocol. Please refer to Appendix G for a summary of MC-OXT.

Oblivious PRF (OPRF) A pseudo-random function (PRF) $F_K(\tau)$, where K is the key and τ is the input, is called oblivious [42] if there is a two-party protocol in which the first party inputs τ, the second inputs K, the first learns the value of $F_K(\tau)$ and the second learns nothing. We make use of the OPRF construction from [34], which is secure against active corruption of the receiver and passive corruption of the party with the key, assuming the one-more Diffie Hellman Assumption (please see Appendix B.1 for details).

4 FED Framework

We use the notion of an *ideal functionality* [26] to specify our requirements from an FED system. An ideal functionality is an (imaginary) trusted entity which interacts with all the players in the system, carrying out various commands from them. A scheme is considered secure if it carries out the same tasks as this trusted entity, with the same secrecy and correctness guarantees.

FED is formulated as a two stage functionality, involving an *initialization* stage and a *query* stage. Fig. 1 depicts the FED functionality schematically. The parties involved are:

- **Data-owners** D_i (for $i = 1, \cdots, m$, say) who are online only during the initialization phase. Each data-owner D_i has an input $Z_i \subseteq \mathcal{W} \times \mathcal{X}$, where for each $(w, x) \in Z_i$, w form the searchable attributes and x the computable values. $Z = \bigcup_i Z_i$ denotes the multi-set union of their inputs.
- **Storage Server** S, which is the only party with a large storage after the initialization stage.
- **Auxiliary Server** A, assumed not to collude with S.
- **Clients** which appear individually during the query phase. A client C has input a query of the form (Q, f) where $Q : \mathcal{W} \to \{0,1\}$ is a search query on the attributes, and f is a computation function on a multi-set of values. It receives in return $f(Q[Z])$ where $Q[Z]$ denotes the *multi-set* consisting of elements in $\mathcal{X}$, with the multiplicity of x being the total multiplicity of elements of the form (w, x) in Z, but restricted to w that are selected by Q; i.e., $\mu_{Q[Z]}(x) = \sum_{w \in \mathcal{W}: Q(w)=1} \mu_Z(w, x)$, where $\mu_R(y)$ denotes the multiplicity of an element y in multi-set R.

We instantiate our framework using protocols that provide security against active corruption of the clients and passive corruption of the other entities, allowing any subset not involving both the servers to collude. Our protocols provide different levels of (acceptable) leakage and are efficient on large scale data. We remark that these protocols indeed satisfy the desiderata outlined in Section 1: The data owners are not required to trust a single central authority or co-ordinate with each other, the clients are oblivious of data owners and of each other, and anonymity of data is maintained modulo the leakage to the storage server and auxiliary sever.

Keyword Search Queries: The major search query families that have received attention in the searchable encryption literature – and also of interest to this work – are "keyword queries."[7] A keyword query is either a predicate about the presence of a single keyword in a record (document), or a boolean formula over such predicates. In terms of the notation above, the searchable attribute for each record is a set of keywords, $w \subseteq \mathcal{K}$ where $\mathcal{K}$ is a given keyword space. That is, $\mathcal{W} = \mathcal{P}(\mathcal{K})$, the power set of $\mathcal{K}$. A basic search query could be a keyword occurrence query of the form Q_τ, for $\tau \in \mathcal{K}$, defined as $Q_\tau(w) = 1$ iff $\tau \in w$. A more complex search query can be specified as a boolean formula over several such keyword occurrence predicates.

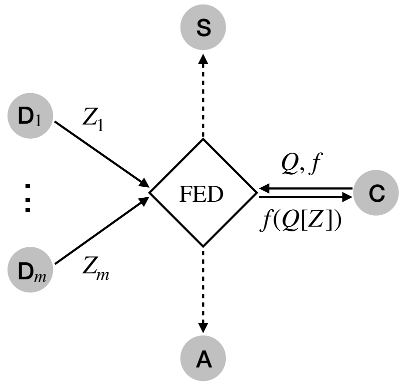

Fig. 1: FED functionality (dotted lines show leakage)

Composite Queries: We shall sometimes allow Q and f to be more general than presented above. Specifically, we allow $Q = (Q_1, \cdots, Q_d)$, where each $Q_i : \mathcal{W} \to \{0,1\}$ and $f = (f_0, f_1, \cdots, f_d)$, where for $i > 0$, f_i are functions on multi-sets of values, and f_0 is a function on a d-tuple; we define $f(Q[Z]) := f_0(f_1(Q_1[Z]), \cdots, f_d(Q_d[Z]))$. (This could be further generalized to recursively allow Q_i and f_i to have the same general structure.)

We note that our framework allows C to specify any query it wants. Authorization of such queries by A is a separate question and we do not discuss it further in this work[8]. Note that for the ideal functionality to be fully specified, we need to describe the leakage functions.

Security Model: We provide provable security guarantees in the Universally Composable (UC) security framework — i.e., in the *real-ideal paradigm* [26], with straight-line, black-box simulation in the presence of an arbitrary environment but with a customized corruption model. In our corruption model all the parties can be passively corrupt, (i.e. honest-but-curious), but in addition the *clients can be malicious or actively corrupt*. Furthermore, the storage server S and the auxiliary server A are assumed to not collude with each other.

Our protocols will be modular: e.g., an FED scheme will be specified in terms of two simpler functionalities, which themselves will be implemented separately. To prove security, we can analyze the protocol while retaining the simpler functionalities as ideal entities, thanks to the composability property of UC security. (which holds also for mixed corruption models involving active and passive corruptions, like the one we use) [11]. While the details of the analysis are routine, the important detail that falls out of this analysis that is significant is the specification of the leakages, which we shall mention along with each of the protocols (along with details on the computational complexity cost). More details on the security analysis of our protocols is given in Section 7.

Notation Summary. For ease of reference, we provide a brief summary of our notation in Table 2.

5 Single Data-Owner Protocols

In this section we introduce a single data-owner version of FED, denoted by sFED, and also construct a sFED scheme. The single data-owner setting is simpler as it avoids having to "mix" data records from different data-owners. Our sFED scheme relies on two other new functionalities we introduce, namely, (single data-owner versions of) Searchably Encrypted Datastore (sSED) and Computably Encrypted Datastore (sCED). We begin by presenting these.

[7] We remark that the concept of searchable encryption has been generalized to more expressive forms of search [54,13,58,14,40,21]. Our general framework applies to all these notions as well.

[8] One simple way of performing such an authorization for A is for A to check the specified queries against a policy stored with it.

5.1 Searchably Encrypted Datastore

Recall that in an FED or sFED scheme, a query has two components – a *search query* Q and a computation function f. The Searchably Encrypted Datastore functionality (SED or sSED) has a similar structure, but supports only the search query; all the records that match the search query are revealed to the storage server S[9]. Jumping ahead, the choice of S to be the party receiving the output rather than the client C is dictated by our plan to use this functionality in protocols for FED and sFED.

The functionality sSED is depicted in Fig. 2: There is a single data-owner D with input $W \subseteq \mathcal{W} \times \mathcal{I}$, where each element in W has a unique identifier $\mathrm{id} \in \mathcal{I}$ as its second coordinate; the output that S receives when a client C inputs Q is the set of identities $Q[W] \subseteq \mathcal{I}$.

In Section 5.5, we shall see that a multi-client version of Symmetric Searchable Encryption (MC-SSE) from [33] can be used to construct an sSED scheme. The main limitations of MC-SSE compared to sSED are that (1) in the former the data-owner D remains online throughout the protocol whereas in the latter D can be online only during the initialization phase, and (2) in the former the output is delivered to both S and C whereas in the latter it must be delivered only to S. In our construction in Section 5.5, we shall leverage the auxiliary server A to meet these additional requirements.

5.2 Computably Encrypted Datastore

The second functionality we introduce – CED, or its single data-owner variant sCED – helps us securely carry out a computation on an already filtered data set. The complexity of this computation will be related to the size of the filtered data rather than the entire contents of the data set.

In sCED, as shown in Fig. 2, a single data-owner D (who is online only during the setup phase) has an input in the form of $X \subseteq \mathcal{I} \times \mathcal{X}$. Later, during the query phase, clients can compute functions on a subset of data. More precisely, a client C can specify a function f from a pre-determined function family, and the storage server S specifies a set $T \subseteq \mathcal{I}$, and C receives $f(\delta_T[X])$ where we define $\delta_T(\mathrm{id}, x) = x$ if $\mathrm{id} \in T$ and $\perp$ otherwise, and $\delta_T[X]$ is the multiset of x values obtained by applying $\delta_T(\mathrm{id}, x)$ to all elements of X.

In Section 5.4, we present protocols for sCED, for various specialized function families, as well as for a general function family.

5.3 sFED Protocol Template

<u>Protocol $\mathtt{sFED}$:</u> This protocol is illustrated in Fig. 2. During the initialization phase, D maps its input Z to a pair (W, X), where $W \subseteq \mathcal{W} \times \mathcal{I}$ and $X \subseteq \mathcal{I} \times \mathcal{X}$ such that $(w, x) \mapsto ((w, \mathrm{id}), (\mathrm{id}, x))$ where id is randomly drawn from (a sufficiently large set) $\mathcal{I}$. Then, in the initialization phase of sFED, the parties D, S and A invoke the initialization phase of sSED, and that of sCED (possibly in parallel). During the query phase of sFED, S, A and C first invoke the query phase of sSED, so that S obtains $T = Q[W]$ as the output; then they invoke the query phase of sCED and C obtains $f(\delta_T[X])$; note that $\delta_T[X] = Q[Z]$ if there are no collisions when elements are drawn from $\mathcal{I}$ to construct (W, X) from Z.

<u>Leakage $\mathcal{L}_{\mathtt{sFED}}$:</u> The protocol $\mathtt{sFED}$ leaks, for every query (Q, f) from C, the set $T = Q[W]$ to S, and in addition provides S and A with the leakage provided by the sSED and sCED functionalities (which depends on how they are instantiated). Note that D chooses ids at random to define W and X and the leakage functions of sSED and sCED are applied to these sets. Leaking T is what is known in the SSE literature as the *access pattern leakage*, i.e. it amounts to leaking the pattern of T over multiple queries. Specifically, since the ids are random, $T_1, \cdots, T_n$ contains only the information provided by the intersection sizes of all combinations of these sets. Formally, this leakage is given by

$$\mathrm{pattern}(T_1, \cdots, T_n) := \{\bigcap_{i \in R} T_i\}_{R \subseteq [n]} \tag{1}$$

[9] This is the access pattern leakage, which is already present in most SSE schemes [20,12].

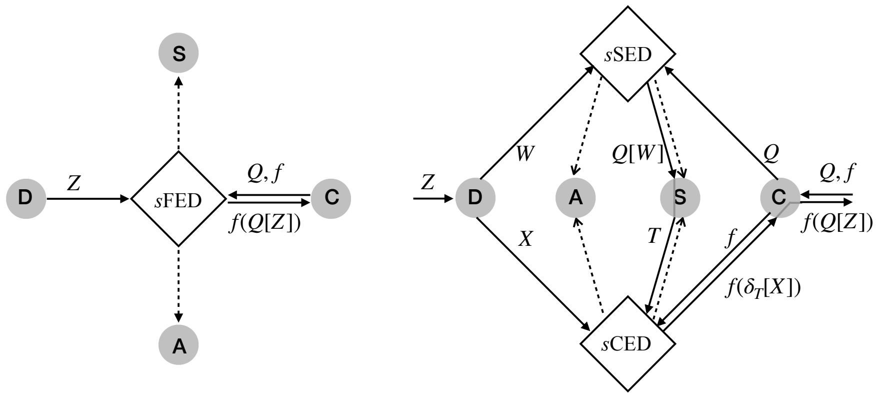

Fig. 2: sFED functionality (left) and the protocol template. The dotted lines indicate leakage. The protocol template is in terms of functionalities sSED and sCED (which are in turn instantiated using separate protocols) and the parties do not communicate to each other outside of (the instantiation of) these two functionalities.

5.4 sCED Protocols

We develop several sCED protocols for computing various functions on data filtered by a search query. All our protocols could be seen as using secure multi-party computation (MPC) techniques to achieve their goals. In particular, these protocols use, as appropriate, secret-sharing schemes, additive homomorphic encryption, and Yao's garbled circuits. The protocols have been adapted to the two-phase, two-server setting of sCED, wherein A can hold only small amounts of data after the initialization phase.

In each of the protocols, D has an input $X \subseteq \mathcal{I} \times \mathcal{X}$ during the initialization phase. It will be convenient to define the set $\mathcal{J} \subseteq \mathcal{I}$ as $\mathcal{J} = \{\text{id} | \exists x \text{ s.t. } (\text{id}, x) \in X\}$. During each query, S has an input $T \subseteq \mathcal{J}$ and C has an input f from the computation function family. The complexity of these protocols are summarized in Appendix D.

Value Retrieval: This is the functionality associated with standard SSE, where the selected values, or documents, are retrieved without any further computation on them. There is a single function in the corresponding computation function family, given by $f(\delta_T[X]) = \delta_T[X]$. When the client C and the data owner D are the same party (as is the case in the simplest version of SSE), this can be implemented in a straightforward fashion using a PRF. Below we give a simple scheme which relies on A to extend this to a setting with (multiple) clients who do not communicate directly with D.
Protocol `sValRet`

- **Initialization Phase:** D picks a PRF key K, sets $\beta_{\text{id}} := x_{\text{id}} \oplus F_K(\text{id})$ and sends $\{(\text{id}, \beta_{\text{id}})\}_{\text{id} \in \mathcal{J}}$ to S and K to A, who store them.
- **Computation Phase:**
  S picks a PRF key K_1, computes a random permutation of T and sends these to A. Additionally, S computes the same permutation of $\{\beta_{\text{id}} \oplus F_{K_1}(\text{id}) \mid \text{id} \in T\}$ and sends this to C. A sends $\{F_K(\text{id}) \oplus F_{K_1}(\text{id}) \mid \text{id} \in T\}$ to C. C outputs $\{a_i \oplus b_i\}_i$, where $\{a_i\}_i$ and $\{b\}_i$ are the messages it received from S and A.
- **Leakage**, $\mathcal{L}_{\texttt{sValRet}}$ **:** On initialization, the ID-set $\mathcal{J}$ is leaked to S. On each query, T is leaked to A.

Recall that, in the overall sFED protocol template, ids will be chosen randomly. Hence leaking $\mathcal{J}$ amounts to leaking only its size $|X|$ (the data can be padded with dummy entries so that instead of $|X|$, only an upper bound on it is leaked), and leaking T amounts to only leaking its *pattern* over multiple queries (called access pattern leakage: see Equation 1).

We briefly sketch the elements in the protocol that help it achieve security. In the initialization phase D secret-shares its data between the two non-colluding servers, so that an adversary corrupting either one learns no information about the data. In the computation phase, C receives freshly randomized secret-shares (using the key K_1) of the answer to its query, with the elements randomly permuted. This is because, if C does not collude with one of the servers it should receive no information other than the multi-set of retrieved values, $f(\delta_T[X])$. In particular, it may not learn whether an id selected by one query gets selected again under another query. The permutation and fresh secret-sharing ensures that its view can be completely simulated just based on the multi-set of retrieved values, $f(\delta_T[X]) = \{x_{\mathrm{id}} | \mathrm{id} \in T\}$. Note that if C colludes with one of the servers, this rerandomization has no effect, but also, in that case, it is allowed to learn T and there is no need for rerandomization.

Summation: The family $\mathcal{F}_{\mathrm{Sum}}$ consists of the single function f such that $f(S) = \sum_{x \in S} x$, where the summation is in a given abelian group which the domain of values is identified with. The following simple and efficient protocol yields a sCED scheme for summation, with A learning only the size and the "pattern" information about the input of S.

Protocol `sSum`

- **Initialization Phase:** D picks a PRF key K, sets $\beta_{\mathrm{id}} := x_{\mathrm{id}} + F_K(\mathrm{id})$ and sends $\{(\mathrm{id}, \beta_{\mathrm{id}})\}_{\mathrm{id} \in \mathcal{J}}$ to S and K to A, who store them.
- **Computation Phase:** S defines the set $R := \{\mathrm{id} \mid \mathrm{id} \in T\}$ and a random value ρ; it sends (ρ, R) to A and $\gamma := \rho + \sum_{\mathrm{id} \in T} \beta_{\mathrm{id}}$ to C. A sends $\delta := \rho + \sum_{\alpha \in R} F_K(\alpha)$ to C. C outputs $\gamma - \delta$.
- **Leakage**, $\mathcal{L}_{\mathtt{sSum}}$ **:** On initialization, the ID-set $\mathcal{J}$ is leaked to S. On each query, T is leaked to A.

This protocol is a natural extension of the Value Retrieval scheme above, with the same initialization phase. The client C receives a fresh additive secret-sharing of the single output value it seeks. The security argument is similar to before; in particular, if C does not collude with either server, its view can be completely simulated from its output without any leakage.

In Appendix C we present an alternate protocol for summation, using additive homomorphic encryption (AHE), which has lower communication complexity and also avoids the leakage of the filtered set T to A. However, owing to the usage of AHE, the protocol is much slower compared to the one above.

Value Retrieval and Summation for vectors: The above protocols can be easily extended to the setting where each value x is a vector $(x_1, \cdots, x_m)$, and the function f acts on a subset of coordinates. C will send the relevant coordinates to A (but not to S). The protocol could be seen as parallel executions of the original protocol, one for each coordinate of interest. The execution is carried out for all coordinates at S, but at the last step, A sends to C only the shares for coordinates included in f. In terms of leakage, the coordinates of interest are leaked to A but not to S. Note that efficiency can be improved at the cost of leaking coordinates of interest to S since then S need not carry out execution for coordinates that are not of interest.

General Functions: Our sCED scheme for general functions could be seen as an adaptation of the CFE scheme of [43] for general functions, despite some important differences between the two models.[10]

A client C who wishes to evaluate a function f sends a circuit representation of f to A. The inputs to this circuit are the values $\{x^{\mathrm{id}}\}_{\mathrm{id} \in T}$ which none of the participants in the query phase (C, A nor S) knows. At a high-level, the idea is that A will construct a garbled circuit for f and sends it to S. For each input bit for this circuit, there are two labels, which will both be encrypted by A using keys that are derived from a master key that the data-owner D gives it (as described below). All these encrypted labels are sent along with the garbled circuit. To evaluate the garbled circuit, S needs to know how to decrypt one out of the two labels corresponding to each input position. To enable the evaluation, D would have provided S with the decryption key for the labels corresponding to each bit of x^{id}, for each id (during the initialization phase). A detailed description of this scheme is given in Fig. 3.

The leakage can be further reduced by allowing C to specify any part of the function as a private *input* to the circuit computing f (rather than being hardwired into the circuit). For each of the wires corresponding

[10] In particular, the model of CFE conflates the client C and the storage server S; also, it does not allow the data-owner D to directly communicate with the auxiliary sever A (resulting in the use of public-key encryption in [43], which we avoid).

to this private input, the two labels will be provided by C to A, and C will send one of these two labels to S (so that neither S nor A by itself learns this input). The formal security proof uses the security of the garbled circuits (similar to the argument in [43]):

Protocol `sCktEval`: A sCED scheme for general functions:
The values computed over (i.e., elements in $\mathcal{X}$) are represented as bit strings of the same size, say $|x| = t$. We shall refer to a Symmetric Key Encryption scheme (SKE) with algorithms (SKE.Gen, SKE.Enc, SKE.Dec) (denoting Key Generation, Encryption and Decryption algorithms respectively).

- **Initialization Phase:** The data-owner D picks a PRF key K. For each $(\text{id}, x^{\text{id}}) \in X$, where $x^{\text{id}} = (x_1^{\text{id}}, \cdots, x_t^{\text{id}})$, it computes $\{\omega_{\text{id},i}, \lambda^0_{\text{id},i}, \lambda^1_{\text{id},i}\}_{i=1}^t$, where $\omega_{\text{id},i} = x_{\text{id},i} \oplus \nu_{\text{id},i}$, with $\nu_{\text{id},i}$ being a pseudorandom bit, and $\lambda^b_{\text{id},i}$ (for $b = 0, 1$) are two encryption keys generated by SKE.Gen, all computed using $F_K(\text{id}, i)$ as the source of randomness. D sends $\{\text{id}, \{\lambda^{x_i^{\text{id}}}_{\text{id},i}, \omega_{\text{id},i}\}_{i=1}^t\}_{\text{id}\in X}$ to S, and it sends K to A, for storing.
- **Computation Phase:** C's input is a function f defined on a multi-set (order of the inputs not being important). We assume that given a number n, a circuit representation of f can be efficiently computed corresponding the input being a multi-set of size n. The storage server S' input is a set $T \subseteq X$.
  1. S sends T to A (randomly permuted). Then, for each id $\in T$, A computes $\{\nu_{\text{id},i}, \lambda^0_{\text{id},i}, \lambda^1_{\text{id},i}\}_{i=1}^t$ using $\{F_K(\text{id}, i)\}_{i=1}^t$.
  2. C sends f to A. Then, A constructs a garbled circuit for f specialized for $|T|$ inputs. The input wires of this circuit are indexed by $\{(\text{id}, i)\}_{\text{id}\in T, i\in[t]}$. Let $(\tau^0_{\text{id},i}, \tau^1_{\text{id},i})$ be the pair of wire-labels used for each such input wire. Then, for each (id, i), and $b \in \{0, 1\}$, it defines the ciphertext
  $$c^b_{\text{id},i} \leftarrow \text{SKE.Enc}_{\lambda^{b\oplus\nu_{\text{id},i}}_{\text{id},i}}(\tau^{b\oplus\nu_{\text{id},i}}_{\text{id},i}).$$
  Note that $c^{\omega_{\text{id},i}}_{\text{id},i}$ is the encryption of $\tau^{x_i^{\text{id}}}_{\text{id},i}$ using the key $\lambda^{x_i^{\text{id}}}_{\text{id},i}$.
  A sends to S $\{(c^0_{\text{id},i}, c^1_{\text{id},i})\}_{\text{id}\in T, i\in[t]}$, along with the garbled circuit *except the output decoding map*. It sends the output decoding map to C.
  3. For each (id, i), S decrypts $c^{\omega_{\text{id},i}}_{\text{id},i}$ using $\lambda^{x_i^{\text{id}}}_{\text{id},i}$, to get $\tau^{x_i^{\text{id}}}_{\text{id},i}$. It uses these labels for the input wires to evaluate the circuit, and obtains the output labels. It sends these output labels to C.
  4. C uses the output decoding map from A, and output labels from S to calculate the output.
- **Leakage, $\mathcal{L}_{\texttt{sCktEval}}$:** On initialization, the ID-set $\mathcal{J}$ is leaked to S. On each query, T and f are leaked to A, and the circuit structure of f (as revealed by the garbled circuit) is leaked to S.
- **Complexity:** $\text{TIME}^{\text{init}}_{\mathsf{D}} = O(|X|)$. Suppose the circuit for f is of size s, and its output has m bits. Then, $\text{TIME}^{\text{query}}_{\mathsf{A}}, \text{TIME}^{\text{query}}_{\mathsf{S}} = O(|T|(s))$; $\text{TIME}^{\text{query}}_{\mathsf{C}} = O(m)$.

Fig. 3: Protocol details for `sCktEval`.

Composite Queries: Recall that a composite query consists of $Q = (Q_1, \cdots, Q_d)$ and $f = (f_0, f_1, \cdots, f_d)$ such that $f(Q[Z]) := f_0(f_1(Q_1[Z]), \cdots, f_d(Q_d[Z]))$. As we shall see in Section 8, composite queries are very useful in practice since they enable confining expensive parts of the computation, for instance using garbled circuits, to smaller size sets obtained by performing inexpensive computation on filtered results. We note that the sSED protocol for non-composite queries directly generalizes to composite queries, by simply running d instances of the original sSED functionality to let S learn $T_i = Q_i[W]$ for each i. But we need to adapt the sCED protocol to avoid revealing each $f_i(Q_i[Z])$ to C.

Towards adapting the above discussed non-composite queries sCED protocols, we observe that in all our protocols, the last step has S and A sending secret shares for the output to C. The reconstruction operation to calculate the final output from these secret shares is simple and efficient. We use this common theme in our protocols to allow calculation of composite queries.

Protocol `sComposite`: We modify our sCED protocols as follows: Instead of S and A sending the shares to C, they carry out a secure 2-party computation of f_0 with each input being secret-shared as above. This can

be implemented for a general f_0 using Yao's garbled circuits and oblivious transfer (OT) [63], or for special functions like linear combinations using standard and simpler protocols.
Leakage, $\mathcal{L}_{\texttt{sComposite}}$ **:** Leakage includes the composed leakage of running d instances to compute $(f_1, \ldots, f_d)$. Also, the function f_0 is leaked to A and S (if f_0 is evaluated using Yao's garbled circuits, only the circuit structure used to evaluate f_0 is leaked to S).

5.5 *s*SED Protocols

In this section, we discuss how to instantiate the building block of *s*SED used in our *s*FED protocols by adapting constructions of symmetric searchable encryption (SSE) [12] from the literature. Though the standard setting for SSE is described in terms of *keyword searches* in a collection of documents, it extends to searches over a table in a relational database as follows: each row in the table is interpreted as a document, consisting of "keywords" of the form (attribute, value), one for each attribute (column) in the table. Then a search query that is specified as a formula over attribute-value equality predicates can be encoded as a formula over keyword predicates.

Recall the Jarecki et al. [33] extension of SSE, which involves a data owner in addition to a client and server. There the data owner is assumed to remain online throughout, as the untrusted server cannot serve client queries itself. In *s*SED, we further separate the roles of the data owner and the query-phase assistant, by introducing an untrusted auxiliary server. This allows the data owner to be present only at the initial phase when the database is constructed. More importantly, in the multiple data owner setting (in which no one data owner can be trusted with access to all the data), it is crucial to avoid relying on any one data owner to control clients' access to the entire database. Finally, in *s*SED, we seek to handle active corruption of clients.

We outline some approaches on how to adapt the MC-OXT protocol of [33] to account for the above requirements of *s*SED. We sketch these in more detail below (Fig. 4). As a lightweight modification, we can simply let the auxiliary server A play the role of the data owner during the query phase, as well as the client in the MC-OXT protocol. This incurs some leakage to A, namely the search queries Q and the search outcome T. Note that since *s*SED is instantiated with random IDs, when used in the *s*FED protocol template (Fig. 2), only the *pattern* of the search outcomes (i.e. the access pattern leakage, as in Equation 1), is revealed to A and S. Also, since many of the *s*CED protocols already leak this information to A, this provides an appropriate level of security for *s*SED protocols to be used in the *s*FED protocol template. We also present two security-efficiency tradeoffs relative to the above protocol (see Fig. 4 for a detailed description of the tradeoffs and the modifications, see Appendix G for a summary of MC-OXT and Appendix A for the notation).

Security - Efficiency Tradeoffs:

- We can avoid leaking the search outcome to A, but only leak an upper bound on the size of the outcome, by using an additive homomorphic encryption instead of plain public-key encryption used in the SSE scheme of [33] to encrypt the ids. This solution could be seen as an instance of onion secret-sharing (Section 1.3) that supports reusability.
- We can retain the original efficiency of the SSE scheme, but slightly increase the leakage to S (only if the keyword query involves a boolean condition over multiple keyword predicates) and avoid leaking the search outcome to A by simply omitting the above mentioned layer of encryption.
- Further, if we start with the OSPIR-OXT protocol in [33] (discussed in Appendix G) that handles actively corrupt clients and prevents leakage of client query to D through the use of OPRF evaluations, while maintaining query authorization through the use of access policies, then we can also avoid leaking the client search query to A. We leave this for future work.

6 FED Protocols

Our FED protocol template is identical to that of the *s*FED protocol, except that the functionalities *s*SED and *s*CED are replaced by the analogous multiple data-owner versions, SED and CED (see Fig. 5).

High Level Overview of query phase of MC-OXT in [33]:

1. C inputs a conjunctive query and sends this to D.
2. D performs computation and sends tokens as authorization to C.
3. C performs additional computation over these tokens and sends them to S.
4. S uses the tokens by C to perform search and returns the encrypted set of document indices to C.
5. C locally decrypts these indices to obtain output of the MC-OXT protocol.
6. C requests the corresponding documents from S.

Modification to an sSED protocol:

– **Initialization phase:** The initialization phase proceeds exactly as in the MC-OXT protocol, except, at the end, the keys for authorizing a query are now sent to A instead of D.
– **Query phase:**
  1. C inputs a conjunctive query and sends this to A.
  2. A performs the computation intended by D and C (above in steps 2,3) and sends tokens to S.
  3. S uses the tokens by A to perform search and returns the encrypted set of document indices to A.
  4. A decrypts the indices to obtain output of the MC-OXT protocol and sends this to S.

Leakage, $\mathcal{L}_{\text{MC-OXT-mod}}$: Leakage to A is equivalent to leakage to D during query phase of MC-OXT. A has an additional leakage of the filtered set of id's and the size of documents matching least frequent keyword. C does not learn the filtered set of id's, the size of the documents matching to least frequent keyword or the pattern of the least frequent keyword. In other words, C does not incur any leakage in this modification. Leakage to S stays same as in MC-OXT.
Complexity: $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(\sum_{w\in\mathcal{K}} |DB(w)|)$; $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|\text{SP}|)$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

Modification using Additive Homomorphic Encryption:
Notation $\langle\langle M\rangle\rangle_{PK}$ denotes additive homomorphic encryption of M using a public-key PK. Let time per encryption be $\text{TIME}_{\text{HomEnc}}$ and decryption be $\text{TIME}_{\text{HomDec}}$.

– **Initialization Phase:**
  1. A and S create a public/secret key-pair $(PK_{\mathsf{A}}, SK_{\mathsf{A}})$ and $(PK_{\mathsf{S}}, SK_{\mathsf{S}})$ for the homomorphic encryption scheme and publish PK_{A} and PK_{S} respectively.
  2. D proceeds exactly similar as in MC-OXT. Whenever D encrypts the database in MC-OXT. It stores an additive homomorphic encryption of id instead of a symmetric key encryption, i.e. $Enc(K, \text{id})$ is replaced by $\langle\langle \text{id}\rangle\rangle_{PK_{\mathsf{A}}}$.
– **Query Phase:**
  1. C inputs a conjunctive query and sends this to A.
  2. A performs the computation intended by D and C (above in steps 2,3) and sends tokens to S.
  3. S uses the tokens by A to perform search and returns the encrypted set of document indices to A, i.e. S receives $\{c_{\text{id}}\}_{\text{id}\in T}$ where $c_{\text{id}} = \langle\langle \text{id}\rangle\rangle_{PK_{\mathsf{A}}}$. For each id, it chooses a random value and adds the random value to the ciphertext. i.e. $\gamma_{\text{id}} = \langle\langle r_{\text{id}}\rangle\rangle_{PK_{\mathsf{A}}} + c_{\text{id}}$ and $\delta_{\text{id}} = \langle\langle -r_{\text{id}}\rangle\rangle_{PK_{\mathsf{S}}}$. S sends $\{(\gamma_{\text{id}}, \delta_{\text{id}})\}_{\text{id}\in T}$ to A.
  4. A decrypts using SK_{A} to reveal $\{(\text{id} + r_{\text{id}}, \delta_{\text{id}})\}_{\text{id}\in T}$. A encrypts using PK_{S} and calculates $\{\eta_{\text{id}}\}_{\text{id}\in T}$ where $\eta_{\text{id}} = \langle\langle \text{id} + r_{\text{id}}\rangle\rangle_{PK_{\mathsf{S}}} + \delta_{\text{id}}$. It permutes this set and sends to S. (Note:$\eta_{\text{id}} = \langle\langle \text{id} + r_{\text{id}}\rangle\rangle_{PK_{\mathsf{S}}} + \langle\langle -r_{\text{id}}\rangle\rangle_{PK_{\mathsf{S}}} = \langle\langle \text{id}\rangle\rangle_{PK_{\mathsf{S}}}$)
  5. S decrypts using SK_{S} to reveal $\{\text{id}\}_{\text{id}\in T}$.

Leakage, $\mathcal{L}_{\text{MC-OXT-mod-adhom}}$: Leakage to A is equivalent to leakage to D during query phase of MC-OXT. A additionally leaks **the cardinality** of the filtered set of id's and the size of documents matching least frequent keyword. C does not incur any leakage in this modification. Leakage to S stays same as in MC-OXT.
Complexity: $\text{TIME}_{DO}^{\text{init}} = O((\sum_{w\in\mathcal{K}} |DB(w)|)\text{TIME}_{\text{HomEnc}})$; $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|\text{SP}| + |T|(\text{TIME}_{\text{HomEnc}} + \text{TIME}_{\text{HomDec}}))$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

Modification by Removing Encryption:

– **Initialization Phase:** D proceeds as in MC-OXT, it stores the plaintext in clear, i.e. $Enc(K, \text{id})$ is replaced by id.
– **Query Phase:**
  1. C inputs a conjunctive query and sends this to A.
  2. A performs the computation intended by D and C (above in steps 2,3) and sends tokens to S.
  3. S uses the tokens by A to perform search and learns the set of filtered indices i.e. $\{\text{id}\}_{\text{id}\in T}$

Leakage, $\mathcal{L}_{\text{MC-OXT-mod-noenc}}$: Leakage to A is equivalent to leakage to D during query phase of MC-OXT. A learns the size of documents matching least frequent keyword and nothing else. C does not incur any leakage in this modification. Leakage to S increases when compared to the leakage of S in MC-OXT. S learns each id in the least frequent keyword and is also able to learn intersection patterns between each id in the least frequent keyword and other queried keywords. However, if performing single keyword search, the leakage in this protocol exactly matches leakage in MC-OXT.
Complexity: $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(\sum_{w\in\mathcal{K}} |DB(w)|)$; $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|\text{SP}|)$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

Fig. 4: Security-Efficiency tradeoffs in modified MC-OXT in [33].

Protocol $\texttt{FED}$: Each data-owner D_i maps its input Z_i to a pair (W_i, X_i), where $W_i \subseteq \mathcal{W} \times \mathcal{I}$ and $X_i \subseteq \mathcal{I} \times \mathcal{X}$ such that $(w, x) \mapsto ((w, \mathrm{id}), (\mathrm{id}, x))$ where id is randomly drawn from (a sufficiently large set) $\mathcal{I}$.[11] After that, all the parties proceed exactly as in $\texttt{sFED}$, but with the parties accessing SED and CED instead of sSED and sCED, and with each data-owner using (W_i, X_i) as its input and $X = \bigcup_i X_i$.

Leakage $\mathcal{L}_{\texttt{FED}}$: The leakage is similar to $\mathcal{L}_{\texttt{sFED}}$, but with leakage from sSED and sCED schemes replaced by those from SED and CED respectively. Specifically, on a client query (Q, f), the leakage consists of the pattern information of the set $T = Q[W]$ to S (also called as access pattern leakage), where $W = \bigcup_i W_i$; also the leakages from SED and CED are provided to S and A.

The main challenge then is in realizing the functionalities SED and CED, for reasonable leakage functions. We present our protocols for realizing these functionalities next.

6.1 Protocol Template for SED

We describe a general protocol template to realize the SED functionality, using access to the sSED functionality. The high-level plan is to let A create a merged database so that it can play the role of D for sSED. However, since we require privacy against A, the merged database should appear shorn of all information (except statistics that we are willing to leak). Hence, during the initialization phase, we not only merge the databases, but also replace the keywords with pseudonyms and keep other associated data encrypted. We use pseudonyms for keywords (rather than encryptions) to support queries: during the query phase, the actual keywords will be mapped to these pseudonyms and revealed to A. These two tasks at the initialization and query phases are formulated as two sub-functionalities — $\mathsf{merge\text{-}map}$ and map— collectively referred to as the functionality mmap, as described below.

Functionality Pair $\mathsf{mmap} = (\mathsf{merge\text{-}map}, \mathsf{map})$

– Functionality $\mathsf{merge\text{-}map}$ takes as inputs W_i from each D_i; it generates a pair of "mapping keys" $K_{\mathsf{map}} = (K_{\mathsf{S}}, K_{\mathsf{A}})$ (independent of its inputs), and creates a "merged-and-mapped" input set, $\hat{W}$. Merging simply refers to computing the (multi-set) union $W = \bigcup_i W_i$; mapping computes the multi-set $\hat{W} = \{(\hat{w}, \hat{\mathrm{id}}) | (w, \mathrm{id}) \in W, \hat{w} = \mathfrak{M}_{K_{\mathsf{map}}}(w),$ and $\hat{\mathrm{id}} = \mathfrak{M}_{K_{\mathsf{map}}}(\mathrm{id})\}$, where the mapping function $\mathfrak{M}$ is to be specified when instantiating this template.[12] It outputs K_{S} to S and $(\hat{W}, K_{\mathsf{A}})$ to A. (Since K_{A} will be stored by A, we require that it be short, independent of the data size.)
– Functionality map takes K_{S} and K_{A} as inputs from S and A respectively, and a query Q from a client C; then it outputs a new query $\hat{Q}$ to C. We shall require that there is a decoding function D such that $Q[W] = \{D_{K_{\mathsf{S}}}(\hat{\mathrm{id}}) | \hat{\mathrm{id}} \in \hat{Q}(\hat{W})\}$, where $\hat{W}$ is as described above.

These functionalities may specify leakages to S and/or A, but not to the data-owners or clients. Note that since map gives an output $\hat{Q}$ to C, we shall require that it can be simulated from Q.

The protocol $\mathsf{SED}^{s\mathrm{SED},\mathsf{mmap}}$ is shown in Fig. 6. In this protocol, sSED is invoked with A playing the role of the data-owner as well. The invocation of $\mathsf{merge\text{-}map}$ is part of the initialization phase and that of map is part of the query phase. As shown in the figure, S uses $D_{K_{\mathsf{S}}}$ (decoding function) to compute its output $Q[W]$ from $\hat{Q}(\hat{W})$.

Note that A does not store any additional information between the two phases (other than what the implementation of sSED requires).

Leakage: Since this protocol delivers the merge-mapped data $\hat{W}$ to A, it leaks certain statistical information about the merged data W (not individual datasets W_i) to A. The exact nature of the leakage depends on the mapping-function $\mathfrak{M}$[13].We note that an adversary corrupting A after the initialization phase will not

[11] $\mathcal{I}$ should be large enough so that we may assume that each (honest) data-owner will use a unique id for each record (w, x), disjoint from the set of IDs used by the others, except with negligible probability.

[12] For notational simplicity, we have specified the mapping using a single function $\mathfrak{M}$ over $\mathcal{W} \cup \mathcal{I}$. Typically, this function acts differently on $\mathcal{W}$ and $\mathcal{I}$, using different parts of the key K_{map}.

[13] This leakage can be avoided by relying on secure two-party computation of a certain function between A and S during initialization, but with high communication costs. Other implications of this leakage include A being able

obtain this leakage, since A deletes this merged data and retains only the short keys.[14] This leakage is also removed if we base our sSED scheme on simpler SSE schemes supporting only single keyword queries (please see Appendix B.3 for this discussion). Leakages from merge-map, map and sSED (the latter on the merged dataset) will also be included in the leakage of this protocol.

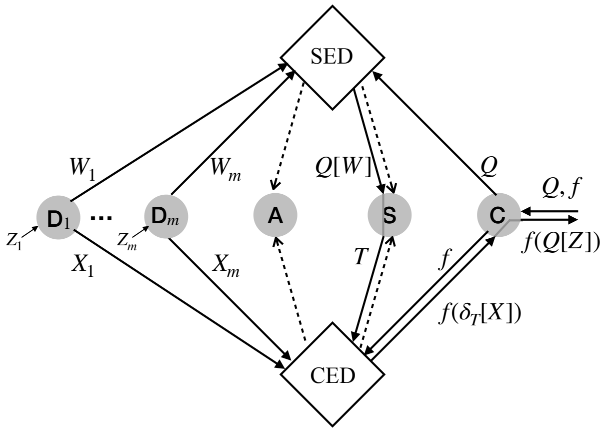

Fig. 5: FED protocol template

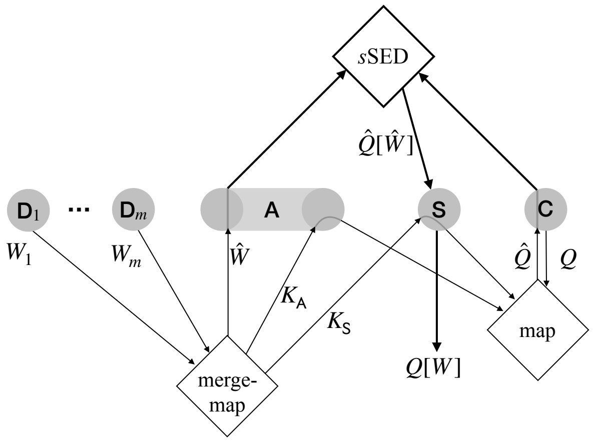

Fig. 6: MDSE protocol template using three functionalities sSED, merge-map and map.

Following the template, to instantiate a FED protocol, we instantiate SED and CED protocols in this section (see Table 3 for a quick summary). To do so, we first introduce the primitive of onion-secret sharing, which we will use in our constructions.

6.2 Onion Secret-Sharing

To illustrate onion secret-sharing, let us first consider the following simplified task: Each sender D_i wants to *share* its message M_i between two servers S and A; that is, it sets $M_i = \sigma_i \oplus \rho_i$, and wants to send σ_i to S and ρ_i to A. While the senders want their messages to get randomly permuted, removing the association of the data with themselves, the association between σ_i and ρ_i needs to be retained. Onion secret-sharing provides a solution to this problem, as follows. Below, let PK_{S} be the public-key of S for a semantically secure public-key encryption scheme (PK_{A} is defined analogously), and let $[\![M]\!]_{PK}$ denote encryption of M using a public-key PK. Now, each D_i sends $[\![(\rho_i, \zeta_i)]\!]_{PK_{\mathsf{S}}}$ to A, where ζ_i is of the form $[\![\sigma_i]\!]_{PK_{\mathsf{A}}}$. A mixes these ciphertexts and forwards them to S, who decrypts them to recover pairs of the form (ρ_i, ζ_i). Now, S reshuffles (or sorts) these pairs, stores ρ_i and sends ζ_i (in the new order); A recovers σ_i from ζ_i (in the same order as ρ_i are maintained by S).

This can be taken further to incorporate additional functionality. As an example of relevance to us, suppose A wants to add a short, private tag to the messages being secret-shared so that the tag persists even after random permutation. Among the messages which were assigned the same tag, A should not be able to link the shares it receives after the permutation to the ones it originally received; S should obtain no information about the tags. Looking ahead, this additional functionality will be required in one of our SED constructions. One solution is for A to add encrypted tags to the data items, and then while permuting the data items, S would *rerandomize* the ciphertexts holding the tags. We present an alternate approach, which

to link the retrieved records with the ones in $\hat{W}$, and further, if it collues with a client, being able to deanonymize the queried keywords.

[14] More formally, assuming adaptive corruption of A after the setup phase (with reliable erasures), the adversary doesn't get this leakage.

does not require additional functionality from the public-key encryption scheme, but instead augments onion secret-sharing with extra functionality:

- D_i holds input M_i and shares it as $M_i = \sigma_i \oplus \rho_i$. It creates a 3-way additive secret sharing of 0 (the all 0's string), as $\alpha_i \oplus \beta_i \oplus \gamma_i = 0$, and sends $(\alpha_i, [\![\beta_i, \rho_i, [\![\gamma_i, \sigma_i]\!]_{PK_\mathsf{A}}]\!]_{PK_\mathsf{S}})$ to A.
- A assigns tags τ_i for each of them, and sends (in sorted order) $(\tau_i \oplus \alpha_i, [\![\beta_i, \rho_i, [\![\gamma_i, \sigma_i]\!]_{PK_\mathsf{A}}]\!]_{PK_\mathsf{S}})$ to S.
- S sends $(\tau_i \oplus \alpha_i \oplus \beta_i, [\![\gamma_i, \sigma_i]\!]_{PK_\mathsf{A}})$ to A, in sorted order; it stores ρ_i (in the same sorted order).
- A recovers $(\tau_i \oplus \alpha_i \oplus \beta_i \oplus \gamma_i, \sigma_i) = (\tau_i, \sigma_i)$.

This allows S and A to receive all the shares (in the same permuted order); S learns nothing about the tags; A cannot associate which shares originated from which D_i, except for what is revealed by the tag. (Even if S or A collude with some D_i, the unlinkability is retained for the remaining D_i). In Section 6.3, we use a variant of the above scheme.

6.3 Instantiating SED Protocol Template

We give two constructions, with different efficiency-security trade-offs. Recall that each data-owner D_i has an input $W_i \subseteq \mathcal{W} \times \mathcal{I}$. In both the solutions, D_i shall use a representation of W_i as a set $\widetilde{W_i} \subseteq \mathcal{K} \times \mathcal{I}$ such that $(w, \mathrm{id}) \in W_i$ iff $w = \{\tau | (\tau, \mathrm{id}) \in \widetilde{W_i}\}$. In the two solutions below, the mapping function $\mathfrak{M}$ maps the keywords differently; but in both the solutions, an identity id that occurs in $\widetilde{W_i}$ is mapped to $\zeta_i^{\mathrm{id}} \leftarrow [\![\mathrm{id}]\!]_{PK_\mathsf{S}}$ (an encryption of id under S's public-key). In one scheme, we use an oblivious pseudorandom function (OPRF) protocol to calculate $\mathfrak{M}$, while the other relies on a secure function evaluation for equality. While an OPRF protocol is more complex than secure equality evaluation, access to OPRF allows us to construct a simpler protocol. On the other hand, this presents a tradeoff in security: suitably efficient OPRF protocols are known only in the heuristic Random Oracle model [33], whereas very efficient secure evaluation of equality is possible in the standard model. The details of the protocols follow. Their complexity is summarized in Appendix D.

Construction `mmap-OPRF` The pair of protocols for the functionalities $\mathsf{merge\text{-}map}$ and map, collectively called `mmap-OPRF`, is shown in Fig. 7. In this solution, the mapping key K_A is empty, and K_S consists of a PRF key K, in addition to a secret-key for a PKE scheme, SK_S. $\mathfrak{M}$ maps each keyword τ using a pseudorandom function with the PRF key K. This is implemented using an oblivious PRF execution with S, in both $\mathsf{merge\text{-}map}$ and map protocols. That is, for $w \subseteq \mathcal{K}$ we have $\mathfrak{M}_{K_{\mathsf{map}}}(w) = \{F_K(\tau) | \tau \in w\}$, where F is a PRF.

Note that `mmap-OPRF` uses two *a priori* bounds (or, includes such bounds as part of leakage) for each data-owner, N_i and L_i such that $|\widetilde{W_i}| \leq N_i$ and $|\mathcal{K}_i| \leq L_i$, where $\mathcal{K}_i = \{\tau | \exists \mathrm{id} \text{ s.t. } (\tau, \mathrm{id}) \in \widetilde{W_i}\}$ is the set of keywords in D_i's data (recall that $\widetilde{W_i}$ is a representation of D_i's data as keyword-identity pairs).

We point out the need for OPRF evaluations. If we allowed S to simply give the key K to every data-owner to carry out the PRF evaluations locally, then, if A colludes with even one data-owner D_i, it will be able to learn the actual keywords in the entire data-set. As described below, using an OPRF considerably improves on this.

Leakage, $\mathcal{L}^{\mathrm{OPRF}}_{\mathrm{mmap}}$: S learns the bounds N_i and L_i for each i, and A learns $\sum_i N_i$; from the map phase, S learns the number of keywords in the query.
We also include in $\mathcal{L}^{\mathrm{OPRF}}_{\mathrm{mmap}}$ what the output $\hat{W}$ to A reveals (being an output, this is not "leakage" for `mmap-OPRF`; but it manifests as leakage in the `SED` protocol that uses this functionality). $\hat{W}$ reveals an anonymized version of the keyword-identity incidence matrix of the merged data,[15] where the actual labels of the rows and columns (i.e., the keywords and the ids) are absent. If A colludes with some data-owners,

[15] This is a 0-1 matrix with 1 in the position indexed by (τ, id) iff $(\tau, \mathrm{id}) \in \bigcup_i \widetilde{W_i}$. The rows and columns may be considered to be ordered randomly.

Protocol `mmap-OPRF`

merge-map:

- **Keys.** S generates a keypair for a CPA-secure PKE scheme, $(PK_{\mathsf{S}}, SK_{\mathsf{S}})$, and similarly A generates $(PK_{\mathsf{A}}, SK_{\mathsf{A}})$. The keys PK_{S} and PK_{A} are published. S also generates a PRF key K.
- **Oblivious Mapping.** Each data-owner, for each $\tau \in \mathcal{K}_i$, D_i engages in an *Oblivious PRF* (OPRF) evaluation protocol with S, in which D_i inputs τ, S inputs K and D_i receives as output $\hat{\tau} := F_K(\tau)$. D_i carries out $L_i - |\mathcal{K}_i|$ more OPRF executions (with an arbitrary $\tau \in \mathcal{K}_i$ as its input) with S.
- **Shuffling.** Each data-owner D_i computes $\zeta_i^{\mathrm{id}} \leftarrow [\![\mathrm{id}]\!]_{PK_{\mathsf{S}}}$, for each id such that there is a pair of the form $(\tau, \mathrm{id}) \in \widetilde{W}_i$. All the data-owners use the help of S to route the set of all pairs $(\hat{\tau}, \zeta_i^{\mathrm{id}})$ to A, as follows:
  - Each D_i, for each (τ, id) computes $\xi_{i,\tau}^{\mathrm{id}} \leftarrow [\![(\hat{\tau}, \zeta_i^{\mathrm{id}})]\!]_{PK_{\mathsf{A}}}$, and sends it to S. Further D_i computes $N_i - |\widetilde{W}_i|$ more ciphertexts of the form $[\![\perp]\!]_{PK_{\mathsf{A}}}$.
  - S collects all such ciphertexts (N_i of them from D_i, for all i), and lexicographically sorts them (or randomly permutes them). The resulting list is sent to A.
  - A decrypts each item in the received list, and discards elements of the form $\perp$ to obtain a set consisting of pairs of the form $(\hat{\tau}, \hat{\mathrm{id}})$, where $\hat{\mathrm{id}} = \zeta_i^{\mathrm{id}}$. $\hat{W}$ is defined as the set of pairs of the form $(\hat{w}, \hat{\mathrm{id}})$ where $\hat{w}$ contains all $\hat{\tau}$ values for each $\hat{\mathrm{id}}$.
- **Outputs.** A's outputs are $\hat{W}$ and an empty K_{A}; S's output is $K_{\mathsf{S}} = (SK_{\mathsf{S}}, K)$.

map: A client C has an input Q which is specified using one or more keywords. For each keyword τ appearing in Q, C engages in an OPRF execution with S, who inputs the key K that is part of K_{S}. The resulting query is output as $\hat{\tau}$.

Fig. 7: Protocol `mmap-OPRF` implementing functionality mmap.

it learns the actual keyword labels of all the corresponding keywords held by the colluding data-owners. As we remarked in the leakage description of SED and Footnote 14, this leakage doesn't occur if A is corrupted after the setup phase or if we use SSE schemes based on single keyword queries for constructing sSED.

Construction `mmap-SFE` Our next protocol avoids OPRF evaluations. The idea here is to allow each data owner to send secret-shared keywords between S and A, and rely on secure function evaluation (SFE) to associate the keywords with pseudonymous handles, so that the same keyword (from different data owners) gets assigned the same handle (but beyond that, the handles are uninformative). Further, neither server should be able to link the keyword shares with the data owner from which it originated, necessitating a shuffle of the outputs. Due to the complexity of the above task, a standard application of SFE will be very expensive. Instead, we present a much more efficient protocol that relies on secure evaluation only for equality checks, with the more complex computations carried out locally by the servers; as a trade-off, we shall incur leakage similar to that of the OPRF-based protocol above. Below, we sketch the ideas behind the protocol, with the formal description in Fig. 8.

Consider two data owners who have shared keywords τ_1 and τ_2 among A and S, so that A has (α_1, α_2) and S has (β_1, β_2), where $\tau_1 = \alpha_1 \oplus \beta_1$ and $\tau_2 = \alpha_2 \oplus \beta_2$. Note that $\tau_1 = \tau_2 \Leftrightarrow \alpha_1 \oplus \beta_1 = \alpha_2 \oplus \beta_2 \Leftrightarrow \alpha_1 \oplus \alpha_2 = \beta_1 \oplus \beta_2$. So, for A to check if $\tau_1 = \tau_2$, A and S can locally compute $\alpha_1 \oplus \alpha_2$ and $\beta_1 \oplus \beta_2$ and use a secure evaluation of the equality function to compare them (with only A learning the result). By comparing all pairs of keywords in this manner, A can identify items with identical keywords and assign them pseudonyms (e.g., small integers), while holding only one share of the keyword.

We note that using *securely pre-computed correlations*, secure evaluation of the equality function can be carried out very efficiently, simply by exchanging a pair of values in a sufficiently large field. The strings to check for equality will be mapped using a collision resistant hash to, say 128 bits, and interpreted as elements in a field, say $\mathbb{F} = GF(2^{128})$. The protocol uses a pre-computed correlation: Alice holds $(a, p) \in \mathbb{F}^2$ and Bob holds $(b, q) \in \mathbb{F}^2$ such that $a + b = pq$ and p, q and one of a, b are uniformly random. Alice sends $u := x + p$ to Bob, where x is her input. Bob replies with $v := q(u - y) - b$, where y is his input. Alice concludes $x = y$ iff $a = v$.

The data shared by the data owners to the servers includes not just keywords, but also the records (document identifiers) associated with each keyword. To minimize the number of comparisons needed, each data owner packs all the records corresponding to a keyword τ into a single list that is secret-shared along with the keyword (rather than share separate (τ, id) pairs). However, after handles have been assigned to the keywords, such lists should be unpacked and shuffled to erase the link between multiple records that came from a single list (and hence the same data owner). Each entry in each list can be tagged by A using the keyword handle assigned to the list, but then the entries need to be shuffled while retaining the tag itself. This is accomplished using the onion secret-sharing technique discussed in Section 1.3. The protocol described there considered a one-bit tag ("marked" or not), but it directly generalizes for bit-strings as tags.

The full protocol involves a few other details: the initial secret sharing of the keywords are delivered to the two servers using onion secret-sharing (so that while adding the tags, A cannot link the data items to the data owners); the number of entries in individual lists for each keyword ; the dummy entries in a list are culled jointly by the two servers. We refer the reader to the detailed description of the protocol in Fig. 8.

Efficiency. We note that this construction uses $O((\sum_i L_i)^2)$ secure equality checks, where L_i is an upper bound on the total number of keywords in D_i's data. In comparison, the previous construction needed $O(\sum_i L_i)$ OPRF evaluations. Even though secure equality checks have a low online cost, when the number of keywords involved is large, the quadratic complexity can offset this advantage. Hence, in Appendix B.2, we provide a mechanism to improve the efficiency of this construction, using a partition of the keyword space into bins.

Leakage, $\mathcal{L}^{\mathrm{SFE}}_{\mathrm{mmap}}$ **:** The leakage includes $\mathcal{L}^{\mathrm{OPRF}}_{\mathrm{mmap}}$ above, with the following additions: during the merge-map phase S learns an upper bound $L \geq |\bigcup_i \mathcal{K}_i|$; from the map phase, A learns the *pattern* of the keywords in a query. During the query phase, the number of keywords in each query are leaked to A.

Please see Section 7 for a detailed analysis of security.

6.4 CED Protocols

Upgrading an sCED protocol to a CED protocol is simpler than the sSED to SED transformation: Since there are no indices to be computed as part of a sCED protocol, we do not need to reconstruct a merge-mapped dataset. Indeed, instead of using sCED functionality as a blackbox, we can directly modify the initialization phase of the sCED protocol. Since the sCED protocols tend to keep all the data records secret-shared between S and A, we need only have the data owners route their data shares to the two servers using onion secret-sharing. We show how the sCED schemes in Section 5.4 are modified in this manner.

Value Retrieval and Summation: Below we describe how the sCED scheme in Section 5.4 for Value Retrieval is adapted into a CED scheme. Adaptation of the Summation scheme is similar.

In the sCED scheme for Value Retrieval, every value x_{id} was secret-shared between A and S as $x_{\mathrm{id}} = \alpha_{\mathrm{id}} \oplus \beta_{\mathrm{id}}$, where $\alpha_{\mathrm{id}} = F_K(\mathrm{id})$, with K being a PRF key that the data-owner and A shared (and S did not have access to). However, this is not viable in a multi data-owner setting, because if S colludes with any one data-owner it will learn this key; alternatively, if each data-owner uses a different key, this would let A collect the different data-items as coming from different data-owners, revealing additional statistics about that data as well as about the answer to queries, beyond what a merged data-set reveals. To prevent such leakage, we delink the key to the data owner and link it to each record, i.e. we generate a new key for each record (x_{id}) that D_i sends. The modified protocol is below. The complexity of this protocol is summarized in Appendix D.

Protocol `ValRet`

- **Modified Initialization Phase:**
  1. S generates key-pairs $(PK_{\mathsf{S}}, SK_{\mathsf{S}})$ for a CPA-secure PKE scheme, and publishes the public-keys.
  2. For each $(\mathrm{id}, x) \in X_i$, D_i picks a PRF key K_{id} and calculates $\beta_{\mathrm{id}} = F_{K_{\mathrm{id}}}(\mathrm{id}) \oplus x_{\mathrm{id}}$ (where F is a PRF with appropriate input and output lengths). D_i sends $\{[\![\delta_{\mathrm{id}}]\!]_{PK_{\mathsf{A}}}\}_{\mathrm{id}}$ to S, where $\delta_{\mathrm{id}} = ([\![K_{\mathrm{id}}]\!]_{PK_{\mathsf{S}}}, (\mathrm{id}, \beta_{\mathrm{id}}))$. (This is possibly padded with random entries, where $\delta_{\mathrm{id}} = \perp$).
  3. S collects all the data from different data owners, sorts them lexicographically and sends them to A.

Protocol `mmap-SFE`

merge-map:

1. **Keys.** S generates a keypair for PKE, $(PK_{\mathsf{S}}, SK_{\mathsf{S}})$, and similarly A generates $(PK_{\mathsf{A}}, SK_{\mathsf{A}})$. A generates a PRF key K. The keys PK_{S} and PK_{A} are published.
2. **Shuffled Sharing.** At the end of this phase, for each (i,τ) such that $\tau \in \mathcal{K}_i$:
   - S holds $(\alpha_{i,\tau}, \mathrm{tag}_{i,\tau})$, where $\alpha_{i,\tau} \in \{0,1\}^\lambda$ is a random string (as long as a keyword), and $\mathrm{tag}_{i,\tau}$ is a unique (random) index for each (i,τ);
   - A holds $(\beta_{i,\tau}, \Gamma_{i,\tau}, \Theta_{i,\tau}, \mathrm{tag}_{i,\tau})$, where:
     - $\beta_{i,\tau} := \tau \oplus \alpha_{i,\tau}$
     - $\Gamma_{i,\tau} := \{\Gamma^{\mathrm{id}}_{i,\tau} | (\tau, \mathrm{id}) \in \widetilde{W}_i\}$ (padded as explained below). Here $\Gamma^{\mathrm{id}}_{i,\tau} := (\xi^{\mathrm{id}}_{i,\tau}, \gamma^{\mathrm{id}}_{i,\tau}, [\![\eta^{\mathrm{id}}_{i,\tau}, [\![\mu^{\mathrm{id}}_{i,\tau}]\!]_{PK_{\mathsf{A}}}]\!]_{PK_{\mathsf{S}}})$ with:
       - $\xi^{\mathrm{id}}_{i,\tau} \leftarrow [\![[\![\zeta^{\mathrm{id}}_i]\!]_{PK_{\mathsf{A}}}]\!]_{PK_{\mathsf{S}}}$ (fresh encryptions for each (i, id, τ)), where $\zeta^{\mathrm{id}}_i \leftarrow [\![\mathrm{id}]\!]_{PK_{\mathsf{S}}}$ (a single encryption per (i, id)),
       - $\gamma^{\mathrm{id}}_{i,\tau}, \eta^{\mathrm{id}}_{i,\tau}, \mu^{\mathrm{id}}_{i,\tau}$ being a random additive secret-sharing of 0^λ (i.e., $\gamma^{\mathrm{id}}_{i,\tau} \oplus \eta^{\mathrm{id}}_{i,\tau} \oplus \mu^{\mathrm{id}}_{i,\tau} = 0^\lambda$).

       $\Gamma_{i,\tau}$ is padded with additional entries of the form $\Gamma^{\perp}_{i,\tau}$ – which is defined identically as $\Gamma^{\mathrm{id}}_{i,\tau}$ except that $\zeta^{\mathrm{id}}_i \leftarrow [\![\perp]\!]_{PK_{\mathsf{S}}}$ – so that $|\Gamma_{i,\tau}|$ is an *a priori* fixed value.
     - $\Theta_{i,\tau} := (\theta_{i,\tau}, [\![\phi_{i,\tau}]\!]_{PK_{\mathsf{S}}})$ for a random $\phi_{i,\tau}$; and $\theta_{i,\tau} := \tau \oplus \phi_{i,\tau}$.

   This phase proceeds as follows:
   - Each D_i, for each $\tau \in \mathcal{K}_i$ picks two random masks $\alpha_{i,\tau}$ and $\phi_{i,\tau}$ and computes $\pi_{i,\tau} := (\beta_{i,\tau}, \Gamma_{i,\tau}, \Theta_{i,\tau})$, as defined above. Then it sends $(\alpha_{i,\tau}, [\![\pi_{i,\tau}]\!]_{PK_{\mathsf{A}}})$ to S. (The number of elements sent by each D_i is padded up to an *a priori* bound, with dummy entries of the form $(\alpha, [\![\perp]\!]_{PK_{\mathsf{A}}})$.)
   - S collects entries of the form $(\alpha_{i,\tau}, \rho_{i,\tau})$ from all D_i, and randomly permutes them. Let $\mathrm{tag}_{i,\tau}$ denote the serial number of these elements in the permuted order. S stores $(\alpha_{i,\tau}, \mathrm{tag}_{i,\tau})$ and communicates $(\rho_{i,\tau}, \mathrm{tag}_{i,\tau})$ to A.
   - A decrypts the entries to obtain $(\pi_{i,\tau}, \mathrm{tag}_{i,\tau})$. (It will discard the entries where $\pi = \perp$.)
3. **Grouping Keywords Using Equality Checks.** At the end of this phase, there is a (hidden) injective mapping $h : \bigcup_i \mathcal{K}_i \to [L]$ (where L equals $|\bigcup_i \mathcal{K}_i|$, or an upper-bound thereof). For each (i,τ) with $\tau \in \mathcal{K}_i$, A will hold a tuple $(h(\tau), \Gamma_{i,\tau}, \Theta_{i,\tau})$.
   - Below we index the entries held by S and A as α_t, π_t etc., t being the tag value. For every two tags, t, t', S and A engage in a 2-party secure equality check protocol to check if $\alpha_t \oplus \alpha_{t'} = \beta_t \oplus \beta_{t'}$. Note that this equality holds iff $\tau_t = \tau_{t'}$. A learns the results.
     (If $\pi_t = \perp$, A uses an arbitrary string instead of β_t in the equality check protocol.)
   - A partitions the set of tags into equivalence classes $T_1, \cdots, T_L$, such that for all $k \in [L]$, and $t, t' \in T_k$, the equality test above was positive. (Hence there is some keyword τ_k corresponding to all the elements in T_k. This implicitly defines the mapping $h : \tau_k \mapsto k$.)
4. **Culling Dummy Entries.** At the end of this phase, for each (i, τ, id) with $(\tau, \mathrm{id}) \in \widetilde{W}_i$, A obtains a tuple $(h(\tau), \zeta^{\mathrm{id}}_i)$.
   - A sends to S (in random order) entries of the form $(h(\tau) \oplus \gamma^{\mathrm{id}}_{i,\tau}, [\![\eta^{\mathrm{id}}_{i,\tau}, [\![\mu^{\mathrm{id}}_{i,\tau}]\!]_{PK_{\mathsf{A}}}]\!]_{PK_{\mathsf{S}}}, \xi^{\mathrm{id}}_{i,\tau})$ where some of the entries may have $\mathrm{id} = \perp$ (in which case, $\xi^{\mathrm{id}}_{i,\tau} = [\![\perp]\!]_{PK_{\mathsf{S}}}$).
   - From each such triple, where the last two items are ciphertexts under PK_{S}, S extracts $(\eta^{\mathrm{id}}_{i,\tau}, [\![\mu^{\mathrm{id}}_{i,\tau}]\!]_{PK_{\mathsf{A}}})$ from the first ciphertext and either $[\![\zeta^{\mathrm{id}}_i]\!]_{PK_{\mathsf{A}}}$ or $\perp$ from the second one. It discards all entries where $\perp$ is obtained from the second ciphertext. It then permutes and sends back entries of the form $(h(\tau) \oplus \gamma^{\mathrm{id}}_{i,\tau} \oplus \eta^{\mathrm{id}}_{i,\tau}, [\![\mu^{\mathrm{id}}_{i,\tau}]\!]_{PK_{\mathsf{A}}}, [\![\zeta^{\mathrm{id}}_i]\!]_{PK_{\mathsf{A}}})$ to A.
   - From each tuple received, A decrypts the second item to obtain $\mu^{\mathrm{id}}_{i,\tau}$, and the third item to receive ζ^{id}_i; combining former with first item, A recovers $h(\tau) \oplus \gamma^{\mathrm{id}}_{i,\tau} \oplus \eta^{\mathrm{id}}_{i,\tau} \oplus \mu^{\mathrm{id}}_{i,\tau} = h(\tau)$.
5. **Sharing the Mapped Keywords.** At the end of this phase:
   - For each (i, τ, id) with $(\tau, \mathrm{id}) \in \widetilde{W}_i$, A obtains a tuple $(h(\tau), \hat{\tau})$ where $\hat{\tau} = F_K(h(\tau))$, and
   - S obtains $\{(k, \sigma_k) | k \in [L]\}$ where, when $k = h(\tau)$, $\sigma_k = \tau \oplus \hat{\tau}$.
   - For each $k \in [L]$, A sets $\hat{\tau}_k = F_K(k)$ Then, it chooses a representative $t^*_k \in T_k$ and sends $(\theta_{t^*_k} \oplus \hat{\tau}_k, [\![\phi_{t^*_k}]\!]_{PK_{\mathsf{S}}})$ to S (sorted by k). (If there are fewer than L equivalence classes, randomly generated θ and ϕ values are used to generate dummy messages.)
   - S receives an ordered list of L pairs $(\delta_k, [\![\phi_k]\!]_{PK_{\mathsf{S}}})$, and it computes $\sigma_k = \delta_k \oplus \phi_k$.
6. **Outputs.** A outputs $K_{\mathsf{A}} = K$ and $\hat{W}$ consisting of elements $(\hat{\tau}, \zeta^{\mathrm{id}}_i)$. S outputs $K_{\mathsf{S}} = (Z, SK_{\mathsf{S}})$, where Z is the set $\{(k, \sigma_k) | k \in [L]\} = \{(h(\tau), \sigma_{h(\tau)}) | \tau \in \bigcup_i \mathcal{K}_i\}$.

map: A client C has an input Q which is specified using one or more keywords. For each keyword τ appearing in Q, C engages in a protocol with S and A to compute $\hat{\tau}$. The protocol is as follows:

- C sends additive secret-shares γ, δ to S and A respectively, where $\gamma \oplus \delta = \tau$.
- For $k = 1, \cdots, L$, S and A carry out an equality check protocol to check if $(\gamma \oplus \sigma_k) = (\delta \oplus F_K(k))$. The output is revealed to A. There will be at most one value k^* such that the equality holds.
- A sends $F_K(k^*)$ to C, who takes it as $\hat{\tau}$. (If no such k^* exists, A sends a random value, or alternately, if permitted, may reveal to C that there is no match.)

Fig. 8: Protocols implementing merge-map and map functionalities for instantiating the template for SED protocols, based on SFE.

4. A receives the data, decrypts it to calculate $\{[\![K_{\mathrm{id}}]\!]_{PK_{\mathsf{S}}}, (\mathrm{id}, \beta_{\mathrm{id}})\}_{\mathrm{id}}$. It then picks a PRF key K. Let $\gamma_{\mathrm{id}} = \beta_{\mathrm{id}} \oplus F_K(\mathrm{id})$. It sends $\{([\![K_{\mathrm{id}}]\!]_{PK_{\mathsf{S}}}, (\mathrm{id}, \gamma_{\mathrm{id}}))\}_{\mathrm{id}}$ to S.

5. S, for each id, decrypts to calculate K_{id}. It defines $\theta_{\mathrm{id}} \leftarrow \gamma_{\mathrm{id}} \oplus F_{K_{\mathrm{id}}}(\mathrm{id})$. (Note that $\theta_{\mathrm{id}} = x_{\mathrm{id}} \oplus F_K(\mathrm{id})$).

6. S stores $\{(\mathrm{id}, \theta_{\mathrm{id}})\}_{\mathrm{id}}$ while A stores K.

The computation phase proceeds exactly as before.

Leakage, $\mathcal{L}_{\texttt{ValRet}}$: The modified initialization phase leaks (an upper bound on) $|X_i|$ for each data-owner to S and the combined ID-set $\mathcal{J}$ to both A and S. (We remark that, in the FED protocol which uses this functionality, the IDs are chosen randomly and hence leaking the ID-set $\mathcal{J}$ only amounts to leaking the size $|X|$ of the combined data set.)

CED **Scheme for General Functions:** The sCED scheme for general functions can be easily adapted to the multi data-owner setting. The only essential modification needed in this case is to route the data from the data-owners to S anonymously (with the help of A). Also, in the modified scheme the data-owners do not provide keys to A to facilitate computation of the keys used to encrypt the labels for each id, but instead the requisite keys will be kept encrypted with S. The modifications are shown in Fig. 9. There is no additional leakage to S compared to the single data-owner version, and A learns an upper bound on the size of the individual ID sets $|\mathcal{J}_i|$ for each data-owner D_i.

Protocol `CktEval`: A CED **scheme for general functions:**

- **Modified Initialization Phase:** Each data-owner D_j, for each $(\mathrm{id}, x^{\mathrm{id}}) \in X_j$, generates $\nu_{\mathrm{id},i}$ and $\lambda^b_{\mathrm{id},i}$ (for $b = 0, 1$) randomly (instead of using a key shared with A). It computes $\zeta_{\mathrm{id}} = \{[\![(\nu_{\mathrm{id},i}, \lambda^0_{\mathrm{id},i}, \lambda^1_{\mathrm{id},i})]\!]_{PK_{\mathsf{A}}}\}_{i=1}^t$, and $\gamma_{\mathrm{id}} = (\mathrm{id}, \zeta_{\mathrm{id}}, \{\lambda^{x_i^{\mathrm{id}}}_{\mathrm{id},i}, \omega_{\mathrm{id},i}\}_{i=1}^t)$, where $\omega_{\mathrm{id},i} = x_{\mathrm{id},i} \oplus \nu_{\mathrm{id},i}$ as before. It sends $\{[\![\gamma_{\mathrm{id}}]\!]_{PK_{\mathsf{S}}}\}_{\mathrm{id}\in\mathcal{J}_j}$ to A who collects them from all the data-owners, sorts them lexicographically, and sends them to S. S decrypts them to obtain $\{\gamma_{\mathrm{id}}\}_{\mathrm{id}\in\mathcal{J}}$, where $\mathcal{J} = \bigcup_j \mathcal{J}_j$.
- **Modification of the Computation Phase:** Step (1) is modified as follows:
  1. S sends $\{\zeta_{\mathrm{id}}\}_{\mathrm{id}\in T}$ to A (randomly permuted), and A decrypts them to obtain, for each $\mathrm{id} \in T$, $\{\nu_{\mathrm{id},i}, \lambda^0_{\mathrm{id},i}, \lambda^1_{\mathrm{id},i}\}_{i=1}^t$.

  The rest of the computation phase proceeds as above.
- **Leakage, $\mathcal{L}_{\texttt{CktEval}}$:** On initialization, the ID-set $\mathcal{J} = \bigcup_j \mathcal{J}_j$ is leaked to S, and the sizes $\{|\mathcal{J}_j|\}_j = \{|X_j|\}_j$ are leaked to A. On each query, T (or rather, its pattern) and f are leaked to A, and the circuit structure of f (as revealed by the garbled circuit) is leaked to S.
- **Complexity:** $\text{TIME}^{\text{init}}_{\mathsf{D}_i} = O(|X_i|\text{TIME}_{\text{Enc}})$; $\text{TIME}^{\text{init}}_{\mathsf{A}} = O(|X|\log|X|)$; $\text{TIME}^{\text{init}}_{\mathsf{S}} = O(|X|\text{TIME}_{\text{Dec}})$. Suppose the circuit for f is of size s, and its output has m bits. Then, $\text{TIME}^{\text{query}}_{\mathsf{A}} = O(|T|(s) + |T|\text{TIME}_{\text{Dec}})$; $\text{TIME}^{\text{query}}_{\mathsf{S}} = O(|T|(s) + |T|\text{TIME}_{\text{Enc}})$; $\text{TIME}^{\text{query}}_{\mathsf{C}} = O(m)$.

Fig. 9: Protocol details for `CktEval`.

7 Analysis

Our security model (Section 4) states that all the parties (D's,A,S) can be passively corrupt and the clients can be actively corrupt. We remark that our protocols can be slightly modified to accommodate actively corrupt data-owners as well, but we do not explore/evaluate this extension in our work. This is because, even given the ideal FED functionality, one actively corrupt data-owner can choose to supply arbitrary data to the database, potentially invalidating the results from the entire database. To adequately handle active corruption of the data-owners, the FED functionality should be augmented with the ability for the servers to enforce policies on the data being collected from the data-owners.

Further we assume that the storage server S and the auxiliary server A do not collude. If the data-owners collude with one of the servers, we still preserve anonymity of data among the remaining data owners. If clients collude with any of the other parties (except both S and A simultaneously), leakage revealed is equivalent to the combined leakages of the involved entities.

7.1 Correctness

We note that correctness of most of our schemes are apparent from the correctness properties of the underlying primitives used to construct them. We briefly discuss them below

- sSED depends on properties of the parent SSE scheme chosen. In case of MC-OXT [33], the protocols are correct, except with negligible probability.
- sCED- `sValRet`, `sSum`, `sCktEval`, `sComposite` are perfectly correct from the perfect correctness of PRF's and garbled circuits. Correctness for `sAlt-Sum` depends on the underlying additive homomorphic scheme used.
- In our FED protocols, onion secret sharing is used on top of the base sFED schemes. Onion secret sharing is correct based on the correctness of the public-key encryption used. SED can run two different protocols - `mmap-OPRF`, `mmap-SFE`.
  Correctness for `mmap-OPRF` depends on the correctness of the underlying OPRF scheme (see Appendix B.1). Correctness for `mmap-SFE` depends on the correctness of our equality checks protocol for secure function evaluation. As the strings are first hashed to an appropriate field, we can have false positives in this protocol. If false positive occur, then during the merge-map functionality, we can group keywords incorrectly and during the map functionality, we can send an incorrect $F_K(k^*)$ to C. This occurs with a very small probability from the security of our collision resistant hash function. From the birthday bound, a false positive occurs with probability (for large output spaces) $e^{\frac{-k(k-1)}{2N}}$ where N is the size of the output space and k is the number of evaluations performed. Assuming 128 bit hashes, $N = 2^{128}$ and an upper bound for $k = (\sum_i L_i)^2 + \mathcal{Q} \times L$ where $\mathcal{Q} = \sum_i |\{\tau : \tau \in Q_i\}|$ is the combined number of keywords in all queries and the rest of the notation is according to Fig. 8. Assuming reasonable parameters for big databases and large amount of queries, the probability here is very low.

7.2 Security

We first provide a detailed security analysis of our SED protocols. The security analysis of the sFED and FED templates are similar to (and somewhat simpler than) that of the SED protocol template discussed below.

We focus on the instantiation of the SED protocol template (Section 6.1) using the components developed in Section 6.3. The analysis of the full SED protocol breaks down into the analysis of the template protocol (Section 6.1) and the protocols for merge-map, map and sSED separately. Note that to enable this modular analysis, it is crucial that these three functionalities are fully specified, including their leakages.

First we analyze merge-map, in Fig. 7. Instead of directly using the random oracle model, we assume an ideal OPRF functionality (which is then UC-securely realized against active corruption of the receiver in the random oracle model Appendix B.1). Given the ideal OPRF functionality, the view of S consists of L_i invocations of OPRF by data-owner D_i, followed by N_i ciphertexts encrypted under PK_{A}. These can be simulated knowing N_i, L_i which are part of the leakage, assuming semantic-security of the public-key encryption scheme. (Note that we assume that all the keywords are encoded into bit-strings of the same length.) The view of A consists of a set of ciphertexts (permuted to disassociate with the data-owners who created it) which can be perfectly simulated knowing only $\sum_i N_i$ (which is part of the leakage) and $\hat{W}$ (which is part of the output). For this, we rely on the semantic security of the public-key encryption scheme. The protocol for map in Fig. 7 is easily seen to be a UC-secure realization of the corresponding functionality as it directly uses the OPRF *functionality.*

Finally, we turn to the analysis of the SED template protocol (see Fig. 6). This relies on the fact that the leakages from SED to A and S are defined to include the corresponding leakages from merge-map, map and

sSED, as well as the information required to simulate the intermediate output (namely $\hat{W}$) that A receives, namely the keyword-identity incidence matrix of the merged data. (Here we rely on the pseudorandomness guarantee of the OPRF to ensure that by learning $\hat{W}$, A learns nothing more than the unlabeled keyword-identity incidence matrix of the merged data, except for the labeling of keywords held by corrupt data-owners.) Also, K_{S} which is output to S is sampled independently of the inputs (and hence perfectly simulated), and the information revealed to S by $\hat{Q}[\hat{W}]$ can be perfectly simulated from $Q[W]$ and K_{S}. In this protocol, if the functionality sSED and merge-map are implemented to handle actively corrupt clients, the overall protocol can also be seen to be admit active corruption of clients. merge-map, as mentioned above, relies on the OPRF protocol for this, and as discussed in Section 5.5, we can modify the searchable encryption protocols from the literature to handle actively corrupt clients.

The analysis of the merge-map and map protocols in Fig. 8 is more tedious, but the simulation is not much more complicated than above. In particular, the ciphertexts received by the servers S and A throughout the protocol (encrypted using the other server's public-key) can be simulated only knowing the number of ciphertexts (by relying on the semantic secuity of public-key encryption). These protocols also rely on a 2-party equality-check protocol, which can be replaced for the purposes of analysis using an ideal equality-check functionality [16]. By carefully inspecting the protocol one can verify that the view of either server (colluding with data-owners or clients, but not with each other) can be entirely simulated using the leakage information specified in Section 6.3. As this analysis is not particularly enlightening, we omit the details and point the reader to the discussion on onion secret-sharing (Section 6.2) for the intuition underlying the construction and the analysis.

8 Implementation and Experimental Results

We show the feasibility and performance of our protocols on realisitically large-scale computations by running tasks representative of *Genome Wide Association Studies (GWAS)*, which is widely used to study associations between genetic variations and major diseases [10].

Implementation/Setup details. The system was implemented in C++11 using open-source libraries (like libPaillier [6], JustGarble [5], Obliv-C [64], NTL [55] and OpenSSL). The MC-OXT protocol of Jarecki et al. [33] was reimplemented for use in our SED protocols. Experiments were performed on a Linux system with 32 GB RAM and a 8-core 3.4GHz Intel i7-6700 CPU. The network consisted of a simulated local area network (simulated using linux tc command), with an average bandwidth of 620 MBps and ping time of 0.1 ms.

Our Functionalities. The specific functions we choose as representative of GWAS were adapted from the *iDash competition* [1] for secure outsourcing of genomic data. Each record in a GWAS database corresponds to an individual, with fields corresponding to demographic and phenotypic attributes (like sex, race etc.), as well as genetic attributes. In a typical query, the demographic and phenotypic attributes are used for filtering , and statistics are computed over the genetic information. In our experiments, the following statistics were computed. Additional details about the implementation/GWAS experimentation are in Appendix E.

- **Minor Allele Frequency (MAF)**: This can be described using the formula $f_0(f_1(Q[Z]), f_2(Q[Z]))$, where f_1 computes $N = |Q[Z]|$, f_2 computes the summation function, and f_0 computes the following formula: $f_0(x, y) = \frac{\min(y, 2x-y)}{2x}$. We implement f_2 using our `sSum` protocol, while f_0 is implemented using our `sComposite` protocol (using GC for f_0).
- **Chi-square analysis (ChiSq)**: Using two search filters for queries Q and Q', χ^2 statistic can be abstractly described using the formula $f_0(f_1(Q[Z]), f_2(Q[Z]), f_3(Q'[Z]), f_4(Q'[Z]))$, where f_1, f_3 are summation functions, f_2 and f_4 compute $|Q[Z]|$ and $|Q'[Z]|$ respectively, and $f_0(a, b, c, d) = \frac{(b+d)(ad-bc)^2}{bd(a+c)((b+d)-(a+c))}$. As previously, f_1, f_3 are implemented using our `sSum` protocol and f_0 implements the above formula using our `sComposite` protocol.
- **Average Hamming Distance**: Hamming distance between 2 genome sequences, often used as a metric for genome comparison[61,48], is defined $f_{x^*}(Q[Z]) = \frac{\sum_{x \in Q[Z]} \Delta(x, x^*)}{|Q[Z]|}$, where Δ stands for the hamming

[16] As collisions occur with negligible probability.

distance of two strings. Here, we consider the entire genotype vector x for each individual (rather than the genotype at a given locus). We implement this using our general functions protocol `sCktEval`.

– **Genome retrieval**: This is a retrieval task, involving retrieval of genomic data at a locus for individuals selected by a search criteria. This is implemented using our `sValRet` protocol.

Dataset and Queries. We used synthetic data inspired by real-world applications [30], with 10,000-100,000 records and 50 SNPs, and with the number of filtered records ranging 2,000-12,000. For Hamming distance, which we implemented using our protocol for general functions (`sCktEval`) and costlier than our other protocols, we used 5,000-25,000 records with 600-3500 filtered records. The experimentation parameters were chosen to showcase the observations described below. Effects of changing the experimental setting (e.g., a WAN instead of a LAN model, bigger datasets and keyword spaces, etc.) are discussed in the full version.

Metrics. Our results are reported over two metrics – the total time taken and the total communication size, across all entities. These metrics are reported separately for the initialization and query phases. Also, the costs incurred by the SED component of the protocols are shown separately, as this could serve as the baseline search cost against which the FED cost can be compared. We performed experiments in both the single (sFED) and multiple data owner (FED) setting, and in the latter case with both the OPRF-based and SFE-based SED protocols.

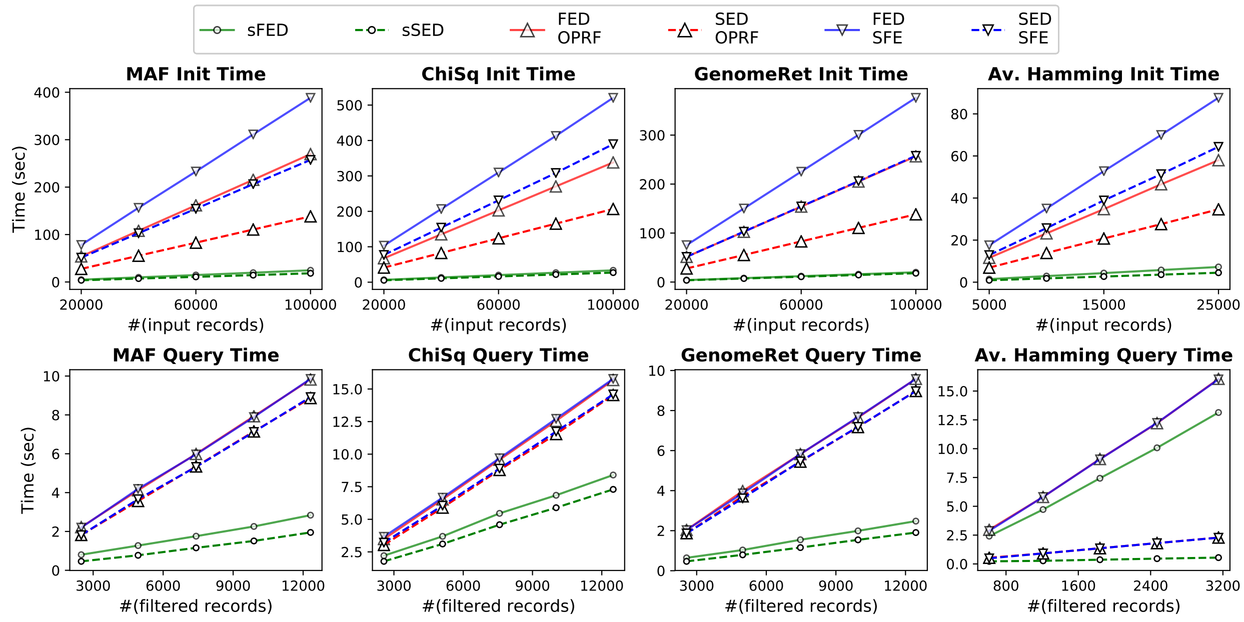

Fig. 10: Initialization and query-phase time plots for our four applications. (See Fig. 13 for communication costs.)

8.1 Observations

Linear in Data. In Fig. 10, we plot the initialization and query times (summed over all parties, with serial computation and LAN communication) for various functions, each one using our different protocols. The experiment runs on a single data-owner (even for FED) and the initialization times are plotted against the total number of records, while query times are plotted against the number of filtered records. As expected, all the times are linear in the number of records. Comparing the different versions of the protocols, we note that for initialization, the multiple data-owner protocols (FED-OPRF and FED-SFE) are slower that the single data-owner protocol (sFED) by a factor of about 10-12x and 15-18x respectively. The overhead in the query phase is a more modest 5x factor.

Efficiency of query phase. From our experimental results in Fig. 10, we observe that the query time for our CED and SED protocols is linearly proportional to the number of filtered records. This is as expected for our CED and sCED protocols, however, for our SED and sSED protocols, the complexity for S is proportional to the number of documents matching the least frequent keyword[17].

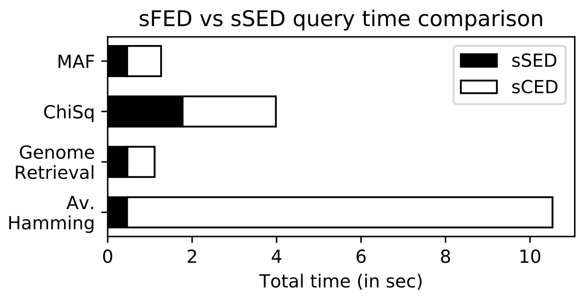

Fig. 11: Break-up of sFED cost.

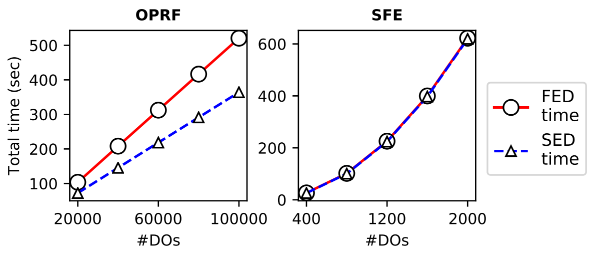

Fig. 12: Scaling of initialization time of FED/SED with increasing number of data-owners (each holding one record)

Comparison to SSE. We show a breakup of our sFED cost between search and compute components in Fig. 11. While the cost for Av. Hamming includes a dataset of 15,000 records and a filtered set of around 1800 records, other protocols involve a dataset of 20,000 records and a filtered set of around 2500 records. As can be observed, the overhead for supporting computation in addition to search ranges between 1.2-1.7x, except for Av. Hamming, which as a result of computing a large function using the protocol `sCktEval`, involves an overhead of 22x. Nevertheless, the computation remains feasible: we evaluate Hamming distance between strings as large as 1.25k bits in 10 seconds.

Scalability with Number of Data-Owners.
The setup phase of our protocols scale well with the number of data-owners participating in the system (see Fig. 12). Each data owner provides one record. As can be seen from the plot, our OPRF based protocol scales to 100,000 data owners in under 10 minutes. The SFE-based protocol (which scales qudratically, but avoids the use of random oracles) scales only up to 2000 owners in a similar time.[18] In our experiments data owners were serviced serially by S, but we estimate a drop of 100-200 sec for our largest benchmarks if we exploited parallelism. Our protocols' performance in the query phase only depend on the total number of records input by the data-owners i.e. a single data owner contributing 10,000 records will have the same query performance as 10,000 data owners contributing 1 record.

Light-weight / Efficient Clients. As is apparent from the complexity analysis of our protocols, our clients are extremely efficient and light-weight, typically only performing computation proportional to size of queries and outputs. Indeed, in all our experiments, client's computation time never exceeded 3 milliseconds.

[17] These results can be seen as a consequence of our dataset construction, where the frequency of any keyword is proportional to the size of the filtered dataset. This restriction helps us simplify the comparison of the FED protocols with their SED components and holds true for datasets in the real world (eg. enron database) [12].

[18] The cost for generating correlated inputs to the server was not included, as this can be done ahead of time even before the setup phase in an efficient manner [37]. Here, the size of the field elements used for the secure equality check was aggressively set to 64 bits, which is adequate to avoid hash collisions for a modest number of data owners and distinct keywords.

8.2 Comparison with Generic MPC Approaches

While MPC techniques cannot be directly deployed in our setting, as the data-owners cannot interact with each other during the query phase, they can be adapted to our 2-server architecture (assuming they do not collude with each other). The two servers could use generic semi-honest secure 2-party computation (2PC) on data that is kept secret-shared between them. However, as we note below, such a solution is significantly less efficient than our constructions.

We focus on the "Genome Retrieval" functionality for a single keyword match with a single data owner, as a minimalistic task for comparison. Note that *for each query*, the servers must engage in a joint computation proportional to the *entire database size*. To ensure a fair comparison, we allow the servers to obtain a similar leakage as our constructions do; this significantly simplifies the functions being securely computed, and reduces the costs.[19]

There are essentially 3 families of generic 2PC protocols which are relevant for the comparison: i) Garbled Circuits (GC) [22], ii) GMW [22] and iii) Fully Homomorphic Encryption (FHE) [2]. For the query phase, GC and GMW suffer heavily in communication cost, since communication between the servers is proportional to the circuit size. For instance, in the case of GC, we can lower bound the communication by restricting to the filtering step in the computation, and then calculating the size of the garbled circuit and oblivious transfers (OT) performed. Concretely, for 100,000 records, this implies a minimum communication size of 576 MB (for details on calculations, please refer to Appendix F). For GMW, for simplicity, we omit the cost of generation of OT correlations (even though, in steady state, they need to be incurred as more queries come in). Then, considering merely 4 bits of communication per AND gate, for the same setting as above, we lower bound the communication during the query phase by 3.6 MB. On the other hand, for the same 100,000 records and a query for filtering around $12,000$ records, our solution incurs only 2.8 MB of communication. For FHE based implementations, the bottleneck is homomorphic computation. A simple lower bound on execution time may be obtained by bounding the number of multiplications performed and considering the cost of a single homomorphic multiplication. For instance, for the smallest size parameters in the Microsoft SEAL homomorphic encryption library [2], the time for a single homomorphic multiplication is the order of milliseconds. Thus a simple estimate for execution time is at least 7 seconds, whereas for the same parameters, we achieve a time of 2.4 seconds. We note that our performance gains increase substantially with increase in the number of records in the database (indeed for 10 million records and a filtered set size of around 500, while we incur 112KB of communication in < 1.7 seconds , GC/GMW require 57.6GB/360MB and FHE is at least of the order of hundreds of seconds). For more details on these calculations, please refer to Appendix F.

9 Conclusion and Future work

In this work, we introduced the primitive of Functionally Encrypted Datastores (FED), which collects data from multiple data-owners, stores it encrypted on an untrusted server, and allows untrusted clients to perform select-and-compute queries on the collected data. There is no leakage of information to clients or data-owners and principled leakage to the servers involved. We provided modular constructions of FED based on constructions of SED and CED, our constructions are summarized in Table 3. Our protocols have leakage and efficiency similar to SSE, but significantly richer functionality. We implemented and tested our protocols on tasks from GWAS, which show that our candidate constructions are quite practical: databases with 100,000 data owners (each contributing a single record) can be securely setup in a few hundred seconds, and a client query that filters up to 12,000 records can be answered in less than 15s with a few MB of total communication in the system.

In the future, we would like to extend our work for broader use cases and stronger security guarantees. We mention some of these ideas below:

[19] To avoid such leakage in this task, a *top-k sorting network* would be included in the computation circuit to collect selected items. A bitonic sorting network for selecting k items out of n has $\Theta(n \log^2 k)$ comparators [53]; selecting a thousand items from a database of a million items (each a few bytes long), would require *billions* of gates.

– *Dynamic Updates*: Our current framework allows for data owners to join only at the beginning of the protocol. Extending this to allow for dynamic addition/updation of data and data-owners is an interesting direction and would increase the scope of applicability of our work.
– *Security Model*: Our current security model (Section 4) assumes the data-owners to only be honest-but-curious and not actively corrupt. Malicious data-owners can introduce data which can force the protocols to abort or arbitrarily corrupt the computation of the data. Extending our current framework to prevent this would be an interesting future direction.
– *Contact tracing application*: Extending our solution to a full-fledged contact tracing application seems to be a useful and non-trivial work item. Our suggested solution keeps in mind the coarsely quantized space and time coordinates for the "filtering step". As an example, given a set of coarsely-quantized space-time coordinates of positive cases (possibly expanded to include spatial neighbors), one can run a filtering query for exact matches in that set. Once the filtering is done, one can carry out additional computation using auxiliary information like finer space/time coordinates. Alternatively, one could also use bluetooth contacts for filtering (each phone could cycle through a set of pseudonyms (keywords) it would emit, and finding which phones came into contact with it would involve keyword matching).
– *Differential privacy*: Techniques from the literature on Differential Privacy could be leveraged to prevent the client-computed statistics to leak information about the identity of the data-owners.

Acknowledgements

We would like to thank our shepherd Marina Blanton and anonymous reviewers for their insightful comments. We would also like to thank Trivikrama Reddy, Kapil Vaidya, Rohan Kadekodi and Deevashwer Rathee for helpful discussions.

S. Agrawal was supported by a DST Indo-Israel grant, a CEFIPRA Indo-French grant, the NSCS National Blockchain project, a Microsoft Research Virtual Centre in Cryptography and the Uchchatar Avishkar Yojana. M. Prabhakaran was supported by SERB with a Ramanujan Fellowship, by DST with an Indo-Israeli grant and by a Microsoft Research Virtual Centre in Cryptography.

References

1. IDASH Privacy & Security Workshop'15. http://www.humangenomeprivacy.org/2015/competition-tasks.html (2015)
2. Microsoft SEAL. https://github.com/Microsoft/SEAL (2019)
3. Baker, J.: Comparing two populations. http://grows.ups.edu/curriculum/Comparing%20Two%20Population1.htm
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of Garbled Circuits. In: CCS (2012)
5. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: IEEE S&P (2013)
6. Bethencourt, J.: Paillier library (2006), http://acsc.cs.utexas.edu/libpaillier/
7. Blakley, G.R., et al.: Safeguarding cryptographic keys. In: Proceedings of the national computer conference (1979)
8. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC (2011)
9. Bonte, C., Makri, E., Ardeshirdavani, A., Simm, J., Moreau, Y., Vercauteren, F.: Privacy-preserving genome-wide association study is practical. IACR Cryptology ePrint Archive (2017)
10. Bush, W.S., Moore, J.H.: Genome-wide association studies. PLoS computational biology (2012)
11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS (2001)
12. Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: CRYPTO (2013)
13. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Asiacrypt (2010)
14. Chase, M., Shen, E.: Substring-searchable symmetric encryption. Privacy Enhancing Technologies (2015)
15. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM (1981)
16. Cho, H., Wu, D.J., Berger, B.: Secure genome-wide association analysis using multiparty computation. Nature biotechnology (2018)

17. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: FOCS (1995)
18. Clarke, G.M., Anderson, C.A., Pettersson, F.H., Cardon, L.R., Morris, A.P., Zondervan, K.T.: Basic statistical analysis in genetic case-control studies. Nature protocols (2011)
19. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: IEEE S&P (2015)
20. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM CCS (2006)
21. Demertzis, I., Papadopoulos, S., Papapetrou, O., Deligiannakis, A., Garofalakis, M.: Practical private range search revisited. In: ACM SIGMOD (2016)
22. Demmler, D., Schneider, T., Zohner, M.: ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS (2015)
23. Fisch, B.A., Vo, B., Krell, F., Kumarasubramanian, A., Kolesnikov, V., Malkin, T., Bellovin, S.M.: Malicious-client security in blind seer: a scalable private dbms. In: IEEE S&P (2015)
24. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC (2009)
25. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC (1987)
26. Goldreich, O.: Foundations of Cryptography: Basic Applications. Cambridge University Press (2004)
27. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F., Sahai, A., Shi, E., Zhou, H.: Multi-input functional encryption. In: EUROCRYPT (2014)
28. Grubbs, P., Ristenpart, T., Shmatikov, V.: Why your encrypted database is not secure. In: Proceedings of the 16th Workshop on Hot Topics in Operating Systems (2017)
29. Hamlin, A., Shelat, A., Weiss, M., Wichs, D.: Multi-key searchable encryption, revisited. In: PKC (2018)
30. Hirokawa, M., Morita, H., Tajima, T., Takahashi, A., Ashikawa, K., Miya, F., Shigemizu, D., Ozaki, K., Sakata, Y., Nakatani, D., et al.: A genome-wide association study identifies plcl2 and ap3d1-dot1l-sf3a2 as new susceptibility loci for myocardial infarction in japanese. European Journal of Human Genetics (2015)
31. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proceedings of the 20th USENIX Conference on Security. SEC'11
32. Jagadeesh, K.A., Wu, D.J., Birgmeier, J.A., Boneh, D., Bejerano, G.: Deriving genomic diagnoses without revealing patient genomes. Science (2017)
33. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: ACM CCS (2013)
34. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P) (2016)
35. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM conference on Computer and communications security (2012)
36. Kamara, S., Raykova, M.: Secure outsourced computation in a multi-tenant cloud. In: IBM Workshop on Cryptography and Security in Clouds (2011)
37. Keller, M., Orsini, E., Scholl, P.: Mascot: Faster malicious arithmetic secure computation with oblivious transfer. Cryptology ePrint Archive, Report 2016/505 (2016), https://eprint.iacr.org/2016/505
38. Kim, M., Lauter, K.: Private genome analysis through homomorphic encryption. BMC medical informatics and decision making (2015)
39. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. In: International Colloquium on Automata, Languages, and Programming (2008)
40. Li, R., Liu, A.X., Wang, A.L., Bruhadeshwar, B.: Fast range query processing with strong privacy protection for cloud computing. Proc. VLDB Endow. (2014)
41. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — a secure two-party computation system. In: In USENIX Security Symposium (2004)
42. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM (2004)
43. Naveed, M., Agrawal, S., Prabhakaran, M., Wang, X., Ayday, E., Hubaux, J.P., Gunter, C.: Controlled functional encryption. In: CCS (2014)
44. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: CCS. ACM (2015)
45. Orencik, C., Selcuk, A., Savas, E., Kantarcioglu, M.: Multi-keyword search over encrypted data with scoring and search pattern obfuscation. International Journal of Information Security (2016)
46. Papadimitriou, A., Bhagwan, R., Chandran, N., Ramjee, R., Haeberlen, A., Singh, H., Modi, A., Badrinarayanan, S.: Big data analytics over encrypted datasets with seabed. In: OSDI (2016)

47. Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A., Bellovin, S.: Blind seer: A scalable private dbms. In: IEEE S&P (2014)
48. Pilcher, C.D., Wong, J.K., Pillai, S.K.: Inferring hiv transmission dynamics from phylogenetic sequence relationships. PLoS medicine (2008)
49. Poddar, R., Boelter, T., Popa, R.A.: Arx: A strongly encrypted database system. IACR Cryptology ePrint Archive (2016)
50. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: In Symposium on Operating Systems Principles (2011)
51. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT (2005)
52. Shamir, A.: How to share a secret. Communications of the ACM (1979)
53. Shanbhag, A., Pirk, H., Madden, S.: Efficient top-k query processing on massively parallel hardware. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1557–1570 (2018)
54. Shi, E., Bethencourt, J., Chan, T.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: IEEE S&P'07 (2007)
55. Shoup, V.: NTL: A library for doing number theory. https://www.shoup.net/ntl/
56. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P, 2000
57. Tkachenko, O., Weinert, C., Schneider, T., Hamacher, K.: Large-scale privacy-preserving statistical computations for distributed genome-wide association studies. In: Asia CCS (2018)
58. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In PVLDB'13 (2013)
59. US Government: http://www.census.gov/ces/dataproducts/index.html
60. Van Rompay, C., Molva, R., Önen, M.: Secure and scalable multi-user searchable encryption. IACR Cryptology ePrint Archive **2018** (2018)
61. Wang, C., Kao, W.H., Hsiao, C.K.: Using hamming distance as information for snp-sets clustering and testing in disease association studies. PloS one (2015)
62. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical private queries on public data. In: {NSDI} 17 (2017)
63. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS (1986)
64. Zahur, S., Evans, D.: Obliv-c: A language for extensible data-oblivious computation. IACR Cryptology ePrint Archive (2015)
65. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (2015)
66. Zelin, A.: https://www.mrs.org.uk/pdf/Andrew_Zelin_presentation.pdf

A Notation

We rely on some standard mathematical notation, like $[n]$ to denote the set $\{1, ...n\}$. Below in Table 2 we summarize additional notation and symbols used across the paper.

B SED protocols: Additional Details

B.1 Actively secure OPRF

In this section, we describe the UC-secure OPRF protocol of [34] which we use in our constructions and which allows active corrution of the receiver and passive corruption of the party with the key.

Let G a cyclic group of prime order m and g be a generator of the group. The receiver, whose input is x, samples $r \in [m]$ uniformly and sends $a = H_0(x)^r$ to the key-holder, where H_0 is a random oracle that maps into the group. The key-holder replies with $b = a^K$, where $K \in [m]$ is the key. The receiver outputs $H_1(x, b^{1/r})$, where H_1 is another random oracle. Formally, then the PRF is described as $f_K(x) = H_1(x, H_0(x)^K)$.

The protocol described above UC-securely implements the ideal OPRF functionality assuming that the (N, Q)-one more Diffie-Hellman Assumption holds for the group. For completeness, we reproduce the assumption (as described in [34]) here, but refer the the reader to Theorem 1 of [34] for further details.

Table 2: Notation index

Notation	Meaning	Notation	Meaning
FED	Functionally Encrypted Datastore	sFED	Single Data-Owner Functionally Encrypted Datastore
SED	Searchably Encrypted Datastore	sSED	Single Data-Owner Searchably Encrypted Datastore
CED	Computably Encrypted Datastore	sCED	Single Data-Owner Computably Encrypted Datastore
S	Storage Server	A	Auxilliary Server
D	Data Owner	C	Client
$\mathcal{W}$	Universe of search-attributes in the data records	W	Input of D to SED functionality; $W \subseteq \mathcal{W} \times \mathcal{I}$
$\mathcal{I}$	Universe of identifiers used for the data records	Z	Input of D to FED functionality; $Z \subseteq \mathcal{W} \times \mathcal{X}$
$\mathcal{X}$	Universe of compute-attributes in the data records	X	Input of D to CED functionality; $X \subseteq \mathcal{I} \times \mathcal{X}$
(Q, f)	Input of C to FED;	T	Set of id's selected by a query; $T \subseteq \mathcal{I}$; $T = Q[W]$
$\mathcal{J}$	The set of unique identifiers for each record in the CED database; $\mathcal{J} \subseteq \mathcal{I}$; $\mathcal{J} = \{\text{id} \mid \exists x \text{ s.t. } (\text{id}, x) \in X\}$	$\text{DB}(w)$	For any $w \in \mathcal{K}$, the set of document indices that contain the keyword w
$\mathcal{L}$	Leakage function	$\mathcal{K}$	Universe of keywords
$[\![M]\!]_{PK}$	Public key encryption of M using a public-key PK	$\text{TIME}^{\text{init}}_{\text{party}}$, $\text{TIME}^{\text{query}}_{\text{party}}$	Time taken during the initialization and query phases, by a party
$\langle\!\langle M \rangle\!\rangle_{PK}$	Additive Homomorphic encryption of M using a public-key PK	$\text{TIME}_{\text{tool}}$	Time taken by a cryptographic tool or operation.
SP	Size Pattern: the number of documents matching the first keyword in a query	$\text{TIME}^{\mathcal{K}}_{\text{OPRF}}$, $\text{TIME}_{\text{OPRF}}$	Time taken in the 2-party OPRF protocol by party with and without a key respectively

(N, Q)*-one more Diffie Hellman Assumption.* For any poly time adversary $\mathcal{A}$,

$$Pr_{k \leftarrow_R \mathbb{Z}_m, g_i \leftarrow_R G}[\mathcal{A}^{(\cdot)^k, DDH(\cdot)}(g, g^k, g_1, g_2, ...g_N) = S]$$

is negligible, where $S = \{(g_{j_s}, g_{j_s}^k) | s = 1, ..., Q+1\}$, Q is the number of $\mathcal{A}$'s queries to the $(\cdot)^k$ oracle, and $j_s \in [N]$ for $s \in [Q+1]$.

B.2 Improving Efficiency

Suppose the keyword space partitions as $\mathcal{K} = \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_\ell$, such that there is an upperbound L_{ij} on $|\mathcal{K}_i \cap \mathcal{B}_j|$, for each data-owner D_i and each bin $\mathcal{B}_j$. Given appropriate statistics on the data domain, such upper bounds can often be derived for easy to compute partitions (e.g., a bin may contain keywords in the English vocabulary that start with a certain set of letters or short prefixes). Then, the protocol can be easily modified so that the equality-checks step (Step 3 in Fig. 8 can be repeated for each bin separately, without having to compare keywords across bins. The total number of equality checks needed, becomes $O(\sum_j (\sum_i L_{ij})^2)$; when ℓ bins are used and if each $L_{ij} = O(L_i/\ell)$, this becomes $O((\sum_i L_i)^2/\ell)$, providing a saving of up to factor ℓ over the original construction. (The assumption that we can take $L_{ij} = O(L_i/\ell)$ would hold only when ℓ is not too large, placing a limit on the efficiency gain possible using binning.)

B.3 Reducing SED Leakage Further

The template used in Section 6.1 reveals the merge-mapped data-set $\hat{W}$ to A. The mapping ensures that the search results $Q[W]$ are not linked to $\hat{W}$, and can be simulated independently given only their pattern information. Nevertheless, $\hat{W}$ itself reveals the entire keyword-identity incidence matrix, as mentioned in

Table 3: Summary of protocols

Protocol Family	Instantiation	Reference	Functionality	Security Assumptions	Leakage(S)	Leakage(A)
sSED	`MD-MC-OXT`	Section 5.5	Search	Non-collusion (A and S), Decision Diffie-Hellman, Random Oracle Model, Secure PRFs	Access pattern, Size Pattern, Equality Pattern, Conditional Intersection Pattern, Total Database Size	Access Pattern, Query
sCED	`sValRet`	Section 5.4	Value Retrieval	Non-collusion (A and S), Secure PRFs	Randomly Chosen IDset ($\mathcal{J}$)	Access Pattern
sCED	`sSum`	Section 5.4	Summation	Non-collusion (A and S), Secure PRFs	Randomly Chosen IDset ($\mathcal{J}$)	Access Pattern
sCED	`sComposite`	Section 5.4	$f_0(f_1(Q_1[Z]), \cdots, f_d(Q_d[Z]))$	Non-collusion (A and S), Oblivious Transfer	Circuit for f_0 (only circuit structure if using Yao's GC) and respective leakages for $f_0, \ldots, f_d$	Circuit for f_0 and respective leakages for $f_1, \ldots, f_d$
sCED	`sCktEval`	Section 5.4	Poly-time computable functions	Non-collusion (A and S), Secure PRFs	Randomly Chosen IDset ($\mathcal{J}$), Circuit Structure for f	Access Pattern, Circuit for f
SED	**`mmap-OPRF`, `MD-MC-OXT`	Section 6.3	Multiple D's + Search	**One More Diffie-Hellman, Public Key Encryption	**Upper bound for frequency of each keyword sent by D_i (called as N_i), Upper bound for number of keywords sent by D_i (called as L_i), Number of keywords in the query	$^{**}\sum_i N_i$, $\hat{W}$ (erased after initialization is complete, can be removed if considering alternate SSE schemes)
SED	**`mmap-SFE`, `MD-MC-OXT`	Section 6.3	Multiple D's + Search	**Random Oracle Model*, Public Key Encryption	**Include leakage from above entry, Upper bound for total number of keywords (L)	**Include leakage from above entry, Number of keywords in the query
CED	**`ValRet`	Section 6.4	Multiple D's + Value Retrieval	**Public Key Encryption	**Upper bound on computing data of D_i (formally $\|X_i\|$)	**Randomly Chosen IDset ($\mathcal{J}$)
CED	**`Sum`	Section 6.4	Multiple D's + Summation	**Public Key Encryption	**Upper bound on computing data of D_i (formally $\|X_i\|$)	**Randomly Chosen IDset ($\mathcal{J}$)
CED	**`CktEval`	Section 6.4	Multiple D's + Poly-time computable functions	**Public Key Encryption	**N/A	**Upper bound on computing data of D_i (formally $\|X_i\|$)

*Random Oracle needed for `MC-OXT-mod`, but not `mmap-SFE`. **In addition to the assumption/leakage in the single data owner analogue.

the leakage description in Section 6.3. This may be acceptable if the statistics of the overall matrix are not private; further, by adding noise in the form of dummy keywords and identities one can further limit the accessible information in $\hat{W}$. In addition, as mentioned in the leakage description of our SED protocols in Section 6.1, the adversary also doesn't obtain this leakage if A is corrupted after the setup phase.

However, there may be scenarios where it would be desirable to almost completely eliminate the leakage from $\hat{W}$ to A. We present a modification of the earlier two constructions which achieves this, as long as the search queries are individual keyword queries (rather than predicates over such keyword queries). Observe that in the case of individual keyword queries, we may map an id used by a data-owner D_i to multiple values $\hat{\text{id}}_\tau$, for each τ such that $(\tau, \text{id}) \in \widetilde{W}_i$. Then, on a keyword query, at most one such value will be recovered (corresponding to the queries keyword); we shall require that on decoding any such value is mapped back to the same id. Then, instead of the (unlabelled) keyword-identity incidence matrix, A learns only the total number of keywords and their "degree distribution" (i.e., the histogram showing, for each n, the number of keywords that appear in n documents in the merged dataset). A does not even learn the total number of ids. For the constructions in Section 6.3 and Section 6.3, this is easily implemented by replacing the ciphertext ζ_i^{id} by $\zeta_{i,\tau}^{\text{id}} \leftarrow [\![\text{id}]\!]_{PK_{\mathsf{S}}}$ (i.e., fresh encryptions for each $(\tau, \text{id}) \in \widetilde{W_i}$).

Therefore, if we base our construction on SSE schemes which deal with single keyword queries (as opposed to schemes which work with predicates over such keywords), we can prevent $\hat{W}$ from leaking the keyword-identity incidence matrix.

C Alternate Protocol for Summation

Our protocol for summation in Section 5.4 uses additive secret sharing to compute $\mathcal{F}_{\text{Sum}}$. In this section we describe an alternative protocol that uses an additive homomorphic scheme to achieve the same functionality. The advantage of using such a scheme is that we can prevent the leakage of the filtered set T to A and reduce the asymptotic communication overheads. However, there is tradeoff in efficiency; the initialization phase is slower as there are additive homomorphic encryptions that need to be computed.

The scheme relies on an additive homomorphic encryption scheme (like Paillier's scheme), which supports rerandomization of ciphertexts (or more specifically, homomorphically adding a random ciphertext (of a random value) to any given ciphertext results in a random ciphertext). The domain of the values $\mathcal{X}$ is identified with a finite group that forms the message space of the homomorphic encryption scheme.

Protocol `sAlt-Sum`

- **Initialization Phase:** D creates a public/secret key-pair (PK, SK) for the homomorphic encryption scheme. It then sends SK to A and $(PK, \{c_{\text{id}}\}_{\text{id}})$ to S, where $c_{\text{id}} \leftarrow \text{HomEnc}_{PK}(x_{\text{id}})$. A and S store the values they receive.
- **Computation Phase:** S first picks a random value $y \in \mathcal{X}$ and computes $\rho \leftarrow \text{HomEnc}_{PK}(y)$; it then computes $c = \rho + \sum_{\text{id} \in T} c_{\text{id}}$, where the summation notations ($+$ and $\sum$) represent the homomorphic addition of ciphertexts supported by the encryption scheme. It sends y to C and c to A. A computes $z \leftarrow \text{HomDec}_{SK}(c)$ and sends z to C, who outputs $z - y$.
- **Leakage:**, $\mathcal{L}_{\texttt{sAlt-Sum}}$ On initialization, the ID-set $\mathcal{J}$ is leaked to S.
- **Complexity:** Let $\text{TIME}_{\text{HomEnc}}$ and $\text{TIME}_{\text{HomDec}}$ be the time for one public key additive homomorphic encryption and decryption respectively. $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(|X|\text{TIME}_{\text{HomEnc}})$; $\text{TIME}_{\mathsf{A}}^{\text{query}} = O(\text{TIME}_{\text{HomDec}})$; $\text{TIME}_{\mathsf{S}}^{\text{query}} = O(|T| + \text{TIME}_{\text{HomEnc}})$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

The main advantage of the scheme is that this prevents the leakage of the set T to A, as S can aggregate the values (under a layer of encryption) without involving A. However, this comes at the cost of efficiency. In particular, the initialization phase here is much slower as it involves public key encryptions for each element in X.

Security against S relies on the semantic security of the homomorphic encryption scheme. In particular, the view of S (during the initialization phase) merely involves ciphertexts. Security against A and C (if not colluding with S) are information-theoretic: during the computation phase A sees only a random ciphertext of a random value, and the view of C consists of a fresh secret-sharing of the final output.

We adapt the protocol above to the multi data-owner setting. The modified scheme has the same computation phase as before, but the initialization phase changes as follows:

Protocol `Alt-Sum`

- **Modified Initialization Phase:** A creates a public/secret key-pair (PK, SK) for the homomorphic encryption scheme, and publishes PK. Each D_i, for each id in its data-set sends $\{[\![(\text{id}, c_{\text{id}})]\!]_{PK_{\mathsf{S}}}\}_{\text{id}}$ to A, where $c_{\text{id}} \leftarrow \text{HomEnc}_{PK}(x_{\text{id}})$ (possibly padded with dummy entries). A gathers these sets from all D_i, sorts them, and sends them to S. S recovers the set $\{\text{id}, c_{\text{id}}\}_{\text{id}}$.
- **Leakage**, $\mathcal{L}_{\texttt{Alt-Sum}}$ The modified initialization phase leaks (an upper bound on) $|X_i|$ for each data-owner to A. (Leakage to S remains the same, namely the ID-set $\mathcal{J}$.)
- **Complexity:** $\text{TIME}_{\mathsf{D}_i}^{\text{init}} = O(|X_i|(\text{TIME}_{\text{HomEnc}} + \text{TIME}_{\text{Enc}}))$; $\text{TIME}_{\mathsf{A}}^{\text{init}} = O(|X| \log |X|)$; $\text{TIME}_{\mathsf{S}}^{\text{init}} = O(|X|\text{TIME}_{\text{Dec}})$; $\text{TIME}_{\mathsf{A}}^{\text{query}} = O(\text{TIME}_{\text{HomDec}})$; $\text{TIME}_{\mathsf{S}}^{\text{query}} = O(|T| + \text{TIME}_{\text{HomEnc}})$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

The protocol relies on the same security analysis as the single data-owner version. Secure routing through A ensures that S does not know about the composition of the data from each data-owner.

D Time Complexity

Here we summarize the time complexity of our various protocols.
sCED **Protocols of Section 5.4**

1. Protocol `sValRet`: $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(|X|)$; $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|T|)$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.
2. Protocol `sSum`: $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(|X|)$; $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|T|)$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.
3. Protocol `sComposite` (for general f_0 with n input bits and circuit of size s): $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(|X|)$; $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(\sum_i^d |T_i| + s + n \cdot \text{TIME}_{\text{OT}})$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

SED **Protocols of Section 6.3**

1. `mmap-OPRF`: Let time taken by a 2-party OPRF protocol by a party with and without a key be $\text{TIME}_{\text{OPRF}}^{K}$ and $\text{TIME}_{\text{OPRF}}$ respectively. Let TIME_{Enc} and TIME_{Dec} be the time for one public key encryption and decryption respectively.
   - $\text{TIME}_{\mathsf{D}_i}^{\text{init}} = O(L_i \cdot \text{TIME}_{\text{OPRF}} + N_i \cdot \text{TIME}_{\text{Enc}})$
   - $\text{TIME}_{\mathsf{A}}^{\text{init}} = O((\sum_i N_i)\text{TIME}_{\text{Dec}})$
   - $\text{TIME}_{\mathsf{S}}^{\text{init}} = O((\sum_i N_i)\log(\sum_i N_i) + (\sum_i L_i)\text{TIME}_{\text{OPRF}}^{K})$
   - $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(|Q|\text{TIME}_{\text{OPRF}})$
   - $\text{TIME}_{\mathsf{S}}^{\text{query}} = O(|Q|\text{TIME}_{\text{OPRF}}^{K})$
2. `mmap-SFE`:
   - $\text{TIME}_{\mathsf{D}_i}^{\text{init}} = O(|\widetilde{W}_i|\text{TIME}_{\text{Enc}})$;
   - $\text{TIME}_{\mathsf{A}}^{\text{init}} = O((\sum_i |\widetilde{W}_i|)\text{TIME}_{\text{Dec}} + (\sum_i L_i)^2 + L \cdot \text{TIME}_{\text{Enc}})$;
   - $\text{TIME}_{\mathsf{S}}^{\text{init}} = O((\sum_i |\widetilde{W}_i|)\log(\sum_i |\widetilde{W}_i|) + (\sum_i L_i)^2 + (\sum_i |\widetilde{W}_i|)\text{TIME}_{\text{Dec}} + L \cdot \text{TIME}_{\text{Dec}})$;
   - $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(L)$.

CED **Protocols of Section 6.4**

1. `ValRet`:
   - $\text{TIME}_{\mathsf{D}_i}^{\text{init}} = O(|X_i| + |X_i|\text{TIME}_{\text{Enc}})$;
   - $\text{TIME}_{\mathsf{S}}^{\text{init}} = O(|X| + |X|\log|X| + |X|\text{TIME}_{\text{Dec}})$;
   - $\text{TIME}_{\mathsf{A}}^{\text{init}} = O(|X| + |X|\text{TIME}_{\text{Dec}})$;
   - $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|T|)$; $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(1)$.

E Implementation / Experimentation: Additional Details

Implementation The following opensource libraries and codebases were used in our implementation.

- OpenSSL library (version 1.0.2) was used for implementing Symmetric Key Encryption (SKE) (based on AES-256) and Public Key Encryption (PKE) (implemented using RSA-OAEP, using hybrid encryption where appropriate). For group operations in the MC-OXT protocol of [33], we used NIST-224p elliptic curves.
- MC-OXT scheme of [33] was reimplemented and ensured that the performance is comparable to that reported in [33].
- Additive Homomorphic scheme was implemented using the Paillier Cryptosystem using the Libpaillier library [6].
- JustGarble [5] was used to implement the CED scheme for general function evaluation.
- Two-party computation for CED for composite queries was implemented using Obliv-C [64]. This provided an implementation of Oblivious Transfer and Yao's Garbled Circuit.[20]
- NTL (Number Theory Library) [55] version 11.3.0 was used to implement the field operations in our construction using secure equality checks.

GWAS Functionalities

[20] Obliv-C provides a high-level interface for implementing a secure 2-party computation protocol. But it is not suited for generating garbled circuits outside the framework of a protocol, for which we relied on JustGarble.

– *Minor Allele Frequency (MAF)*: In what is called a biallelic setting, each locus of interest has one of three genotypes – say AA, BB or AB (corresponding to two "alleles" A and B, and two chromosomes). Given a set of N individuals (selected using a search filter) and a locus of interest, the total allele counts are defined as $n_A = 2n_{AA} + n_{AB}$ and $n_B = 2n_{BB} + n_{AB}$, where n_{AA}, n_{BB} and n_{AB} are the number of individuals of the three genotypes. Then, MAF for that locus is defined [38] as the real number

$$\frac{min(n_A, n_B)}{n_A + n_B} = \frac{min(n_A, 2N - n_A)}{2N}.$$

– *Chi-square analysis (ChiSq)*: To quantify the association between an allele and a disease, often a chi-square (χ^2) statistic defined below is employed [18]. Two groups of individuals – case and control groups – are selected using two search filters Q and Q' (with and without a disease). Defining (n_A, n_B) and (n'_A, n'_B) as above, but for Q and Q' respectively, the χ^2 statistic [38,3] can be formulated as

$$\chi^2 = \frac{(N + N')(n_A N' - n'_A N)^2}{NN'(n_A + n'_A)((N + N') - (n_A + n'_A))}$$

where $N = (n_A + n_B)$ and $N' = (n'_A + n'_B)$. Computing this quantity corresponds to a composite query $f_0(f_1(Q[Z]), f_2(Q[Z]), f_3(Q'[Z]), f_4(Q'[Z]))$, where f_1, f_2, f_3, f_4 compute n_A, N, n'_A, N' respectively, and f_0 implements the above formula (up to finite precision).

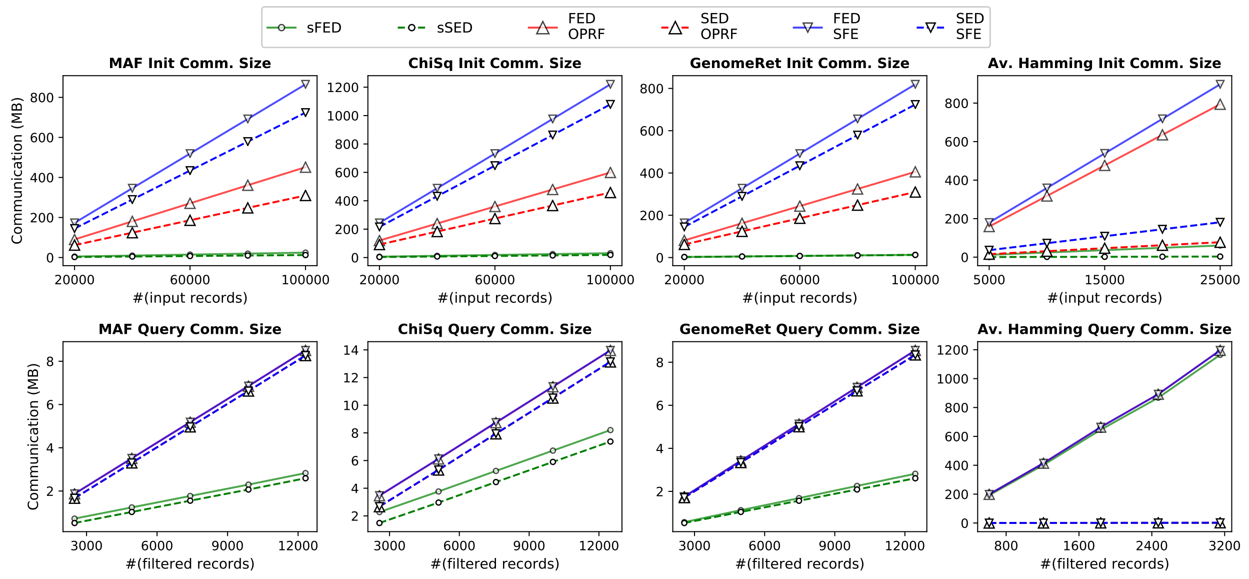

Fig. 13: Total communication across all entities in the initialization/query phases for single/multi data owner (contrasting two SED protocols in Section 6.4, OPRF and SFE) for our different applications.

F Comparison with generic MPC approaches: more details

In this section we provide further details on comparing our work with more generic MPC approaches. As stated in Section 8.2, we consider 3 families of generic 2PC protocols - Garbled Circuits (GC), GMW and Fully Homomorphic Encryption (FHE). Since GC and GMW are known to be communication heavy, we lower bound their communication for purposes of this comparison, while for FHE we lower bound its computation time. More details of each family of protocol is discussed below. For convinience, we let N denote the number of records, q the size of the field over which the search attributes are defined and λ the security parameter.

- **Garbled Circuits**: We lower bound our communication by restricting to only the filtering step of the computation and estimating the size of the Garbled Circuits and OTs performed. For OTs, we make use of OT extension as used in ABY [22]. For our current task of Genome Retrievel, to ensure fair comparison, we let one of servers learn the indices of the filtered records. A lower bound on the number of AND gates and input wires in the resultant circuit is then given by qN. Assuming 2λ bits of communication/AND gate for the garbled circuit, λ bits of communication/input wire from garbler to evaluator and 2λ bits for the OT performed using OT extension between the garbler and the evaluator, we lower bound the total communication by $5qN\lambda$. For $N = 100,000$ records, $q = 72$ (as used in our implementation) and $\lambda = 128$, this gives us a total communication for 576 MB. For the same parameters, but for 10 million records, this is 57.6GB. In comparison, for $100,000$ records and filtered set size of $12,000$ our solution only incurs 2.8MB of communication, while for 10 million records and filtered set size of 500, we incur only 112KB of communication.
- **GMW**: We use the latest optimizations from ABY [22] to lower bound the communication here. However, we omit the cost associated with generating the OT correlations, even though in steady state where clients keep sending queries, generating such OT correlations would nevertheless incur some cost. Then considering only 4 bits of communication per AND gate of the circuit and a crude lower bound on the number of AND gates as qN, we estimate a total communication of $4qN$. For $N = 100,000$ records and $q = 72$ (as used in our implementation), this gives us 3.6MB, while for the same parameters and filtered set size of $12,000$ records we incur only 2.8MB. On the other hand, for 10 million records and filtered set size of 500, while we incur only 112KB of communication, this estimate lower bounds the communication by 360MB.
- **FHE**: Here we assume that one of the servers involved stores the decryption key, while the other one stores all the encrypted data. During the query phase, the client sends an encrypted query to the server, which performs homomorphic computation to compute the result and finally decyrpt it using the key stored with the other server. To lower bound the dominant homomorphic computation cost here, we lower bound the number of multiplications performed by restricting to only the search part of the computation and lower-bounding the time of each ciphertext-ciphertext multiplication by taking the time for such multiplications from Microsoft SEAL [2]. For the latter, we use the smallest set of parameteres supported and for this we found the time to be order of milliseconds, with a polynomial ring of degree 1024. For filtering, we assume binary circuits and lower bound the number of multiplications by qN. Assuming a packing factor given by the degree of the polynomial ring, we lower bound the computation time by $(qN/1024) * 1\text{msec}$. For $N = 100,000$ records and $q = 72$ (as used in our implementation), this accounts to a computation cost of at least 7 seconds. For 10 million records, this is at least of the order of hundreds of seconds. On the other hand, for $100,000$ records and filtered set size of $12,000$ we achieve a time of 2.4 seconds, while for 10 million records and filtered set size of 500 we achieve < 1.7seconds.

G Overview of SSE in literature

G.1 MC-OXT

In this section, we give a brief overview of the multi-client SSE scheme (MC-OXT) of [33]. In this setting, we have a data owner D who processes a large database DB to produce an encrypted database EDB and a master key MSK, of which it sends EDB to the server S. The model allows for multiple clients C, who may request D for tokens corresponding to certain queries Q, which D constructs using its MSK. It provides the client with a key corresponding to Q, along with a signature so that S may verify that C is authorized to make this query. The client transforms the received token into search tokens by doing computation and sends these search tokens with their corresponding signatures to S, who verifies the signature, and executes a search protocol. Finally, it obtains encrypted id values from the server who is in possession of the decryption key.

The MC-OXT protocol of [33] is designed to support arbitrary Boolean queries but for ease of exposition here, we will consider the special case where the client makes conjunctive queries of the form $w_1 \wedge w_2 \wedge \ldots \wedge w_n$ where $\forall\ i, w_i \in \mathcal{K}$ (w_i are keywords in the document, and conjunction means that we wish to query the document with all the keywords $w_1, \ldots, w_n$). The least fequent word, w_1 (say) is known as s-word, while the remaining words $w_2, \ldots, w_n$ are called x-words. It is assumed that the data owner maintains information that lets it compute the s-word for any query of a client.

Setup of MC-OXT: Pick PRFs F_τ and F_p with the appropriate ranges, and key K_S for F_τ and two keys K_X, K_I for F_p. The keys K_S and K_X are used to generate pseudorandom handles of keywords $w \in \mathcal{K}$, denoted by $\mathsf{strap} = F_\tau(K_S, w)$ and $\mathsf{xtrap} = g^{F_p(K_X, w)}$ respectively. The key K_I is used to generate a pseudorandom handle of document indices $\mathrm{id} \in \mathsf{DB}$ as $\mathsf{xind} = F_p(K_I, \mathrm{id})$. Next, the protocol computes $\mathsf{xtag} = g^{F_p(K_X, w)\cdot \mathsf{xind}}$ for all "valid" (w, id) pairs, i.e. pairs such that the document id contains the keyword w, and adds xtag to construct a set called XSet. The idea of constructing a set such as XSet is that if we can query for all documents containing the word w_1 (our s-word) using a list T_{w_1}. Then we can check $\forall\ \mathrm{id} \in T_{w_1}$ if id contains the keyword w_2 by computing if $g^{F_p(K_X, w_2)\cdot F_p(K_I, \mathrm{id})} \in \mathsf{XSet}$. To construct the list T_{w_1} securely, we construct another set called TSet. More formally, for $w \in \mathcal{K}$, the data owner generates a fresh pair of keys (K_z, K_e) using $\mathsf{strap}(w)$ and encrypts list of id's containing keyword w by $e = \mathsf{Enc}(K_e, \mathrm{id})$. Next, for $c \in [|T_w|]$, it blinds the $|T_w|$ representatives $\mathsf{xind}_1, \ldots, \mathsf{xind}_{T_w}$ as follows: set $z_c = F_p(K_z, c)$ and $y = \mathsf{xind} \cdot {z_c}^{-1}$. It stores (e, y) in $\mathsf{TSet}(w)$. Note that (e, y) are constructed using keys that are specific to s-word w. It then chooses a key K_M for $\mathsf{AuthEnc}$ - an authenticated encryption scheme, and this key is shared with the server. The keys $(K_S, K_X, K_I, K_T, K_M)$ are retained by D as the master secret.

Query of MC-OXT: When the client requests a token for the query $w_1 \wedge w_2 \wedge \ldots \wedge w_n$, the data owner computes the s-word, computes stag using K_T, strap using K_S as well as $\mathsf{xtrap}_2, \ldots, \mathsf{xtrap}_n$ corresponding to $w_2, \ldots, w_n$ using K_X. Next, it blinds the xtrap values as $\mathsf{bxtrap}_i = g^{F_p(K_X, w_i)\cdot \rho_i}$ where ρ_i are chosen randomly from $\mathbb{Z}_p^*$. Then it creates an authenticated encryption $\mathsf{env} \leftarrow \mathsf{AuthEnc}(K_M, (\mathsf{stag}, \rho_2, \ldots, \rho_n))$ and returns $SK = (\mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, \ldots, \mathsf{bxtoken}_n)$ to the client.

The client uses SK to construct search tokens as follows. It uses strap to compute K_z and for $c \in [T_w]$ it computes $z_c = F_p(K_z, c)$. Now, it sets $\mathsf{bxtoken}[c, i] = \mathsf{bxtrap}_i^{z_c}$ and sends env along with all the $\mathsf{bxtoken}[c, i]$ to the server. The server verifies the authenticity of env using K_M, and decrypts it to find stag and $\rho_2, \ldots, \rho_n$. It uses stag to obtain $\mathsf{TSet}(w_1)$ and a list of $\{(e, y)\}$. Then for $i \in \{2, \ldots, n\}$ and $\forall y \in \mathsf{TSet}(w_1)$, it checks if $\mathsf{bxtoken}[c, i]^{y/\rho_i} \in \mathsf{XSet}$ (Thus checking if document id $\in T_{w_1}$ contains the keyword w_2 or not). If the membership succeeds, it retrieves the corresponding encrypted document index e and sends it to the client who decrypts it and requests the corresponding document from the server.

Leakage, $\mathcal{L}_{\text{MC-OXT}}$ **:** We summarize the leakage analysis for all queries that are non-adaptive and are for conjunctions of two keywords only. Please refer to [33] for the complete analysis.

Let Q be a sequence of non-adaptive 2-conjunction queries, where $\forall i,\ Q[i] = (s, x)$ where an individual query is a 2-term conjunction $s[i] \wedge x[i]$ which we write as $Q[i] = (s[i], x[i])$. For $w \in \mathcal{K}$, let $\mathrm{DB}(w)$ be the document indices that contain the keyword w. The leakage to S is shown below:

– $\mathcal{L}_{\text{TSet}}, \mathcal{L}_{\text{XSet}}$, this is given by, $\Sigma_{w\in\mathcal{K}}|\mathrm{DB}(w)|$. The total number of appearances of keywords in documents. (Leaked during the setup phase of the protocol).
– The results pattern(RP) of the queries, which are the indices of documents matching the entire conjunction. Formally, a vector of size $|Q|$ with $\mathrm{RP[i]} = \mathrm{DB}(s[i]) \cap \mathrm{DB}(x[i])$.
– The number of documents matching the first keyword in the query, denoted by Size Pattern(SP). Formally, a vector of size $|Q|$ with $\mathrm{SP} = |\mathrm{DB}(s[i])|$.
– The number of queries which have equal first terms (s-words), denoted by Equality Pattern.
– The conditional intersection pattern(IP), which is formally a $|Q|$ by $|Q|$ table defined by $\mathrm{IP}[i, j] = \mathrm{DB}(s[i]) \cap \mathrm{DB}(s[j])$, if ($i \neq j$ and $x[i] = x[j]$), otherwise ϕ.

Additionaly, we leak Q to D and SP to C. Complexity of the protocol is D (Setup) - $O(\sum_{w\in\mathcal{K}}|DB(w)|)$; D, C and S (Query) - $O(|\mathrm{SP}|)$. Please refer to [33] for a complete formal analysis.

G.2 OSPIR-OXT

[33] also describes the OSPIR-OXT protocol, which seeks to reduce the leakage to D using OPRF evaluations and policy based access controls. The primary change to enable this in the MC-OXT protocol is to change the query authorization process. To prevent leakage of the query Q to D and still enable authorization, a two-party OPRF evaluation takes place between D and C. We refer to [33] for the complete details of this protocol.