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

Shweta Agrawal [*†]
IIT Madras

Rachit Garg [*‡]
UT Austin

Nishant Kumar [*†]
Microsoft Research, India

Manoj Prabhakaran [*]
IIT Bombay

***Abstract*—We introduce the notion of a *Functionally Encrypted Datastore* which collects data 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 security and performance profile is similar to that of conventional *searchable encryption* systems, while the functionality we offer is significantly richer. 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 use our protocols to model computations relevant to the *Genome Wide Association Studies* such as Minor Allele Frequency (MAF), Chi-square analysis and Hamming Distance.**

I. Introduction

Given the importance of cloud computing today, enabling controlled computation on large 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) [82], [28], fully homomorphic encryption (FHE) [26] or functional encryption (FE) [71], [7] can provide strong security guarantees, their computational and communication requirements are incompatible with most of the realistic applications today. At the other end are efficient tools like CryptDB [67], Monomi [76], Seabed [62] and Arx [65], which add a lightweight encryption layer under the hood of conventional database queries, but, as we discuss later, offer limited security guarantees and do not support collaborative databases. 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 Searchable Encryption (starting with [75], [21]), and Controlled Functional Encryption [58] – they offer limited functionality.

In this work, we propose a new solution – called *Functionally Encrypted Datastore* (FED) – that can be used to implement a secure cloud-based data store that collects data 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 security and performance profile is similar to that of conventional *searchable encryption* systems, while the functionality we offer is significantly richer.

Some Motivating Scenarios.: There are many scenarios today when data is collected from many users in a centralized database and made available to other users for querying. In such situations, the individuals whose data has been collected should be considered the actual data-owners, who are providing their data with an expectation that it will be used only for certain well-specified purposes. One such example is that of census data: governments use census data for administrative purposes, and also provide restricted access to select researchers for select purposes [77], [85]. Another scenario involves private corporations who can collect large amounts of information about their customers, which could be legitimately useful in improving their customer service. A third example, and the one that we use in our experimental analysis, is offered by genomic studies. *Genome Wide Association Studies (GWAS)* look into entire genomes across different individuals to discover associations between genetic variants and particular diseases or traits [9], [53], [55].

However, in all such scenarios, individuals' privacy is vulnerable and the absence of fine grained control can lead to significant problems. This was illustrated in an incident involving Ipsos Mori, a British marketing research firm, who offered to share data pertaining to 27 million mobile phone users with the police [44]. While they insisted that they will only release safe aggregated data which their users had acquiesced to [60], the deal was shelved under public pressure. Similarly, GWAS studies involve sharing highly sensitive information about the individuals with several researchers in different parts of the world, and the individuals may have little control over how their data will be used in future. Indeed, as evidenced by the case of the Havasupai tribe against the Arizona State University [34], [39], the researchers from the university collected genetic data for studying links to diabetes and later used it for other sensitive subjects like schizophrenia, migration and inbreeding, without the consent of the individuals or the community who contributed the data.

Addressing the Challenge.: To support the above scenarios, we require a datastore that provides strong security guarantees and practical efficiency while offering flexible functionality — multiple data owners, and support for expressive computation queries. To accommodate multiple data owners in a flexible and scalable manner calls for several features:

[*] Author names sorted alphabetically.
[†] Part of work done at IIT Delhi.
[‡] Work done primarily at IIT Madras.

- *No coordination among data-owners.* The data-owners should not need to trust, or even be aware of each other.
- *Untrusted clients.* The clients should not need to hold any state or key material (except as may be needed to authenticate themselves to the servers, in a higher-level application). This allows opening up the datastore to a large set of clients without requiring to trust them.
- *Data-owners oblivious of the clients.* The data-owners should not need to be aware of the clients who will query the datastore in the future, and in particular, cannot provide them with any keys. Further, the data-owners need not be online when the queries arrive from the clients.
- *Anonymously collaborative database.* The association of the encrypted data items with their data-owners should be hidden from all the parties in the system.

A Trust Assumption: The careful reader may have noted a basic conflict in the requirements described above: if the data-owners are oblivious of the clients and are not online when clients access the datastore, then the datastore must be fully operable by the servers. However, this allows the (untrusted) servers to freely access the collected data (e.g., by emulating the role of clients to make as many accesses as they wish), and learn arbitrary information. *Resolving this tension necessitates a trust assumption*: we require that there are multiple *non-colluding* servers. In practice, such an assumption could be based on the growing availability of cloud computing services from competing service providers. We seek a solution that uses multiple servers minimally – there should be only one server with large storage (who stores all the encrypted data), and a single *auxiliary server* (who may store some key material). Either server by itself can be corrupt, but they will be assumed not to collude with each other We remark that the model of non-colluding servers has had great impact in theoretical and practical literature in the past, including multi-prover interactive proof systems [4], multi-server Private Information Retrieval [19], Distributed Oblivious RAM [54], CFE[1] [58] and in Searchable Encryption [61], [66].

Our Approach: We observe that all the motivating scenarios discussed above seek to compute some function on data selected according to some criteria: the government may wish to compute the average income of some community of people in a given state, private corporations may wish to study correlations between usage of some facility and gender/income, medical researchers may wish to study connections between certain diseases and certain genetic traits. In each of these scenarios, the computation proceeds by *selecting* a fraction of the data according to some filtering criteria and then performing some *computation* on selected data – this *select-and-compute* functionality is then the focus of our work. Another way to view this functionality is as typical (relational) database operations on a single table.

In more detail, we consider queries to be specified as a pair (Q, f) where Q is a *filtering predicate* which selects some subset of the dataset 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. This would offer substantial efficiency gains in most practical applications: for instance, in a census-like database for a country, a selection query for a zip-code would result in a small fraction of all the records to be chosen.

Finally, we seek a level of cryptographic security and efficiency that is typical of existing searchable encryption works (e.g., [59], [49]). Here, the security guarantees are formulated in the form of an ideal functionality (following the Universally Composable security framework) and well-defined leakage functions specified as part of this functionality. These solutions are highly scalable and capable of handling large databases [43], [59], [10], [21], with the query-time complexity scaling with the number of selected rows (as discussed above). However, we note that these works are restricted to a single data owner and only search queries. In our work, we shall set a similar security-efficiency combination as the goal, while requiring a much richer functionality.

A. *Our Results*

We introduce 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.

Apart from introducing the notion of FED 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).
- To support multiple data owners, our SED construction introduces a *versatile collection of techniques* under the umbrella of "onion secret-sharing". These techniques leverage ideas from "onion encryption", originally used in anonymous routing and mixed nets [70], [18].
- We provide *instantiations* of this framework for evaluating arbitrary functions, as well as important special classes of functions, the latter more efficiently than the former.
- We demonstrate the utility and practicality of our protocols based on *extensive experimentation*, involving realistic statistical analysis tasks for genomic studies.

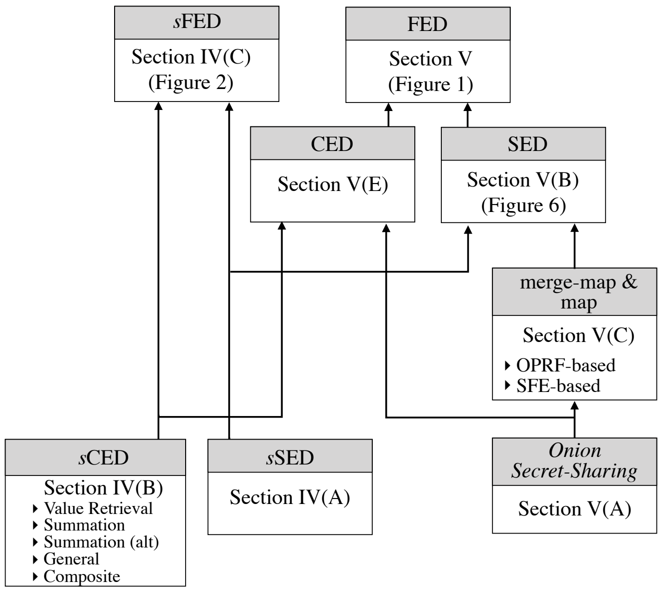
Along the way we give constructions for "single data-owner" versions of FED, SED and CED (denoted by sFED, sSED and sCED, respectively), which are simpler and more efficient, albeit offering no support for multiple data-owners. These may be of independent interest.

B. *Overview of Constructions*

We present several modular constructions of FED (and the single data-owner version sFED), which can be instantiated in multiple ways, by plugging in different implementations of its components. As discussed above, we identify two simpler primitives, Searchably Encrypted Datastore (SED) and Computably Encrypted Datastore (CED), and show how they can

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

be securely dovetailed into a construction of FED. Following is a roadmap to the constructions in this work.



The starting point of our constructions are single data-owner versions *s*SED and *s*CED, shown at the bottom of the above map. We show that these components can be implemented by leveraging constructions from the literature, namely, the Multi-Client Symmetric Searchable Encryption (MC-SSE) scheme due to Jarecki et al. [40] and Controlled Functional Encryption (CFE) due to Naveed et al. [58]. The search query family supported by our *s*SED constructions are the same as in [40]. For *s*CED, we support a few specialized functions, as well as a general function family. The primitives *s*SED and *s*CED are of independent interest, and also they can be combined to yield a single data-owner version of FED (called *s*FED).

To upgrade *s*SED and *s*CED 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. 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 (see Section V-A for an overview). In the case of CED, merging essentially consists of a multi-set union. But in the case of *s*SED, merging entails merging "search indices" on individual data sets into a search index for the combined data set. Since the data-owners do not trust (or are even aware of) each other, this operation must be implemented via the servers in an "oblivious" fashion, and onion secret-sharing techniques alone are 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).

C. Related Work

Below, we contrast our notion of Functionally Encrypted Datastores with prior work in terms of various features. Furthermore, in Table I, we give a comparison of our work with different related works. As is evident from the table, our work represents a different trade-off between functionality, security and query-phase efficiency.

• *Multiple data owners*: We allow encrypted data to be collected from multiple data-owners. Prior works that allow this include multiparty computation [82], [28], multi-input functional encryption [30], multi-key fully homomorphic encryption [52] and controlled functional encryption (CFE) [58], all of which suffer costs proportional to the entire dataset. A different line of work based on multi-key/multi-user Searchable Encryption [69], [68], [33], [79] allows data to be shared from multiple data owners to clients for searching of single keywords, but unlike this 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) [21], [11], [10], [63], [59], [23], [8] and its extensions (including Structured Encryption [14], [42], Queryable Encryption [16] etc.) 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 [67], Monomi [76], Seabed [62] and Arx [65] do allow full-fledged search-*and*-compute (searches are attribute based rather than keyword queries). But they incur higher leakage to the server and argue security in the "snapshot attacker model" which has been criticized for being unrealistically weak [32]. 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 queries and outputs, typically independent of database size. In comparison, the CryptDB family of constructions [67] do not execute all the computational operations fully at the server, but instead require the clients to download certain intermediate results, decrypt them and perform the remaining computation. Similarly, the CFE scheme of [58] 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 [33], [79], which assume clients to be fixed at the start of the protocol and the data owner to know about all such clients. We do, however, rely on the availability of two non-colluding servers, which, as mentioned earlier, is a necessary assumption.

TABLE I: 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 [67]	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 [62]	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 [65]	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 [11]	N	Y	NA	N	N	Passive	Passive	Passive	-	$O(t_1 k)$	$O(t_1 k)$
OSPIR [40]	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 [63], [25]	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 [58]	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 [33], [79]	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 COL$_1$ $= \alpha_1$ AND COL$_2$ $= \alpha_2 \cdots$ 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 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.

II. Preliminaries

We rely on some standard mathematical notations, like $[n]$ to denote the set $\{1, ...n\}$. Furthermore, to aid the reader, a detailed index of the notation used in our paper is provided in Table III. 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) [82], which has been extensively studied [82], [2], [36], [56] and has moreover been optimized for performance by a host of recent works [48], [47], [51], [83], [24], [37].

Secret sharing: Secret sharing [72], [6] 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 *authorised* 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 [75], [27], [31], [13], [21], [15], [43], a (single) client offloads storage of encrypted data to an untrusted server such that it may later perform search queries on this data. SSE permits some principled leakage, namely access and query patterns of the data, to the server. A multi-client version of SSE was proposed by Jarecki et al. [40]: here, the data owner is separate from (and does not trust) the clients and remains online throughout the protocol. For the interested reader, we provide a summary of MC-OXT in Appendix C.

Oblivious PRF (OPRF): A pseudo-random function (PRF) $F_K(\tau)$, where K is the key and τ is the input, is called oblivious [57] 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 [41], 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 D1 for details).

III. FED Framework

We use the notion of an *ideal functionality* [29] 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

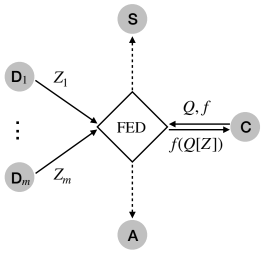


Fig. 1: The FED functionality. The dotted lines indicate leakage from functionality. Note that we do not allow any leakage to the data owners or the clients.

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. Figure 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.

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."[2] 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.

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.)

[2]We remark that the concept of searchable encryption has been generalized to more expressive forms of search [73], [15], [76], [17], [50], [22]. Our general framework applies to all these notions as well.

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[3]. 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* [29], 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.

IV. 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.

A. 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[4]. 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 Figure 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 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 IV-E, we shall see that a multi-client version of Symmetric Searchable Encryption (MC-SSE) from [40] 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

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

[4]The leakage of documents that match the search query to S exist in most SSE systems when a user requests these matched documents from S [11].

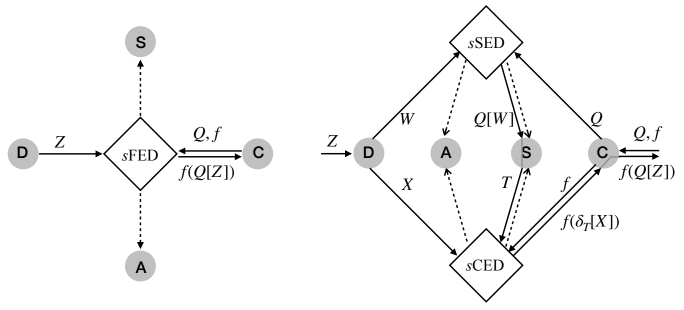


Fig. 2: sFED functionality (left) and the protocol template. The dotted lines indicate leakage. The protocol template is in terms of functionalities sSED and sCED (which are in turn instantiated using separate protocols) and the parties do not communicate to each other outside of (the instantiation of)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

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 IV-E, we shall leverage the auxiliary server A to meet these additional requirements.

B. 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 Figure 2, a single data-owner D (who stays online only during the initialization 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 $\delta_T(\text{id}, x) = x$ iff $\text{id} \in T$, and $\delta_T[X]$ is the multiset of x values obtained by applying $\delta_T(\text{id}, x)$ to all elements of X.

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

C. sFED Protocol Template

Protocol `sFED-templ`**:** This protocol is illustrated in Figure 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 $(w, x) \mapsto ((w, \text{id}), (\text{id}, x))$ 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.

Leakage $\mathcal{L}_{\texttt{sFED-templ}}$: The protocol `sFED-templ` 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. Also, as ids are random, leaking T amounts to only leaking its *pattern* over multiple queries: specifically, $T_1, \cdots, T_n$ contains only the information provided by the intersection sizes of various combinations of these sets. Formally, this leakage is given by

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

D. sCED Protocols

Next, we present protocols for different computation functionalities. 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 E.

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, and defines $\beta_{\text{id}} := x_{\text{id}} \oplus F_K(\text{id})$ and sends $\{(\text{id}, \beta_{\text{id}})\}_{\text{id} \in X}$ to S and K to A, who store them.
- **Computation Phase:** S sends T (randomly permuted) and a fresh PRF key K_1 to A; it also sends $\{\beta_{\text{id}} \oplus F_{K_1}(\text{id}) \mid \text{id} \in T\}$ (under the same permutation) 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 (in the same order).
- **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 (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, $f(\delta_T[X]) = \{x_{\text{id}} | \text{id} \in T\}$. 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 $\mathcal{F}_{\text{Sum}}$ 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, and defines $\beta_{\text{id}} := x_{\text{id}} + F_K(\text{id})$ and sends $\{(\text{id}, \beta_{\text{id}})\}_{\text{id}}$ to S and K to A, who store them.
- **Computation Phase:** S defines the set $R := \{\text{id} \mid \text{id} \in T\}$ and a random value ρ; it sends (ρ, R) to A and $\gamma := \rho + \sum_{\text{id} \in T} \beta_{\text{id}}$ to C. A sends $\delta := \rho + \sum_{\alpha \in R} F_K(\alpha)$ to C. C outputs $\gamma - \delta$.
- **Leakage**, $\mathcal{L}_{\texttt{sSum}}$ **:** 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 B 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 attributes. 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: We provide a sCED scheme for general functions. This scheme makes use of garbled circuits and encryption, and can be seen as an adaptation of the CFE scheme from [58] [5] . Despite the fact that garbled circuits have been optimised for performance in a series of recent works [48], [47], [51], [83], [24], [37], garbled circuits incur high communication complexity and make this protocol much less efficient that our other protocols.

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 $\{x^{\text{id}}\}_{\text{id} \in T}$ 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 Figure 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 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 [58]):

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 VII, 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) [82], or for special functions like linear combinations using simpler protocols. For e.g., if f_0 is addition, and the inputs are additively secret-shared in the same group, S and A each can simply add the shares they have locally, and send the result to C.

Leakage, $\mathcal{L}_{\texttt{sComposite}}$ **:** Leakage includes the composed leakage of running d instances of $(f_1, \ldots, f_d)$. The function f_0

[5] 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 [58], which we avoid).

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 $(\mathrm{id}, x^{\mathrm{id}}) \in X$, where $x^{\mathrm{id}} = (x_1^{\mathrm{id}}, \cdots, x_t^{\mathrm{id}})$, it computes $\{\omega_{\mathrm{id},i}, \lambda^0_{\mathrm{id},i}, \lambda^1_{\mathrm{id},i}\}_{i=1}^t$, where $\omega_{\mathrm{id},i} = x_{\mathrm{id},i} \oplus \nu_{\mathrm{id},i}$, with $\nu_{\mathrm{id},i}$ being a pseudorandom bit, and $\lambda^b_{\mathrm{id},i}$ (for $b = 0, 1$) are two encryption keys generated by SKE.Gen, all computed using $F_K(\mathrm{id}, i)$ as the source of randomness. D sends $\{\mathrm{id}, \{\lambda^{x_i^{\mathrm{id}}}_{\mathrm{id},i}, \omega_{\mathrm{id},i}\}_{i=1}^t\}_{\mathrm{id}\in X}$ 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 $\mathrm{id} \in T$, A computes $\{\nu_{\mathrm{id},i}, \lambda^0_{\mathrm{id},i}, \lambda^1_{\mathrm{id},i}\}_{i=1}^t$ using $\{F_K(\mathrm{id}, i)\}_{i=1}^t$.
  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 $\{(\mathrm{id}, i)\}_{\mathrm{id}\in T, i\in[t]}$. Let $(\tau^0_{\mathrm{id},i}, \tau^1_{\mathrm{id},i})$ 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
     $$c^b_{\mathrm{id},i} \leftarrow \mathrm{SKE.Enc}_{\lambda^{b\oplus\nu_{\mathrm{id},i}}_{\mathrm{id},i}}(\tau^{b\oplus\nu_{\mathrm{id},i}}_{\mathrm{id},i}).$$
     Note that $c^{\omega_{\mathrm{id},i}}_{\mathrm{id},i}$ is the encryption of $\tau^{x_i^{\mathrm{id}}}_{\mathrm{id},i}$ using the key $\lambda^{x_i^{\mathrm{id}}}_{\mathrm{id},i}$.
     A sends to S $\{(c^0_{\mathrm{id},i}, c^1_{\mathrm{id},i})\}_{\mathrm{id}\in T, i\in[t]}$, 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 $c^{\omega_{\mathrm{id},i}}_{\mathrm{id},i}$ using $\lambda^{x_i^{\mathrm{id}}}_{\mathrm{id},i}$, to get $\tau^{x_i^{\mathrm{id}}}_{\mathrm{id},i}$. 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.

Fig. 3: A sCED scheme for general functions

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).

E. sSED *Protocols*

In this section, we discuss how to instantiate the building block of sSED used in our sFED protocols by adapting constructions of symmetric searchable encryption (SSE) [11] 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 predicatescan be encoded as a formula over keyword predicates.

Recall the Jarecki et al. [40] extension of SSE (Section II), 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 sSED, 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. Another difference between sSED and SSE is that the latter reveals the search outcome (the IDs of the rows that match the search) to the client; in sSED, we seek to reveal this to S, but it cannot be revealed to the client (as the clients are not provided with any leakage). Finally, in sSED, we seek to handle active corruption of clients.

We outline some approaches on how to adapt the MC-OXT protocol of [40] to account for the above requirements of sSED. We sketch these in more detail in Figure 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 and the search outcome (the former can be avoided, please see Figure 4). Note that since sSED is instantiated with random IDs, when used in the sFED protocol template (Figure 2), only the *pattern* of the search outcomes, as in Equation 1, rather than the search outcomes themselves are revealed to A, as well as to S. Also since many of the sCED protocols already leak this information to A, this provides an appropriate level of security for sSED protocols to be used in the sFED protocol template.

Many existing SSE schemes and their multi-client versions in literature can be modified and used in our setting of SED. We describe one such instantiation by using the MC-OXT scheme of [40] for the family of keyword search queries. We provide the core ideas of their scheme in Appendix C, explain our modifications in Figure 4 and use this for our implementation. Our modifications do not require prior knowledge of the scheme and can be understood by a high level idea of the entities involved and the functionality expected. Our techniques for modifying MC-OXT scheme can be extended to other SSE schemes as well. Please see Figure 4 for a high-level idea of the protocol flow in MC-OXT, our modifications as well as security-efficiency tradeoffs.

Security - Efficiency Tradeoffs We also present multiple security-efficiency tradeoffs relative to the above protocol (see Figure 4 for a detailed description of the 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 [40] to encrypt the ids. This solution could be seen as an instance of onion secret-sharing (see Section V-A) 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 [40] that handles actively corrupt clients, then we can avoid leaking the search queries to A.

V. FED Protocols

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

Protocol `FED-templ`: 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) $\mathcal{I}$.[6] After that, all the parties proceed exactly as in `sFED-templ`, 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 $\mathcal{L}_{\texttt{FED-templ}}$: The leakage is similar to $\mathcal{L}_{\texttt{sFED-templ}}$ defined earlier, but with leakage from sSED and sCED schemes replaced by those from SED and CED. Specifically, on a client query (Q, f), the leakage consists of the set (or equivalently, the pattern information of) $T = Q[W]$ to S, 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. To do so, we first introduce the primitive of onion-secret sharing, which we will use in our constructions.

A. *Onion Secret-Sharing*

In going from single data-owner schemes to multi data-owner schemes, we seek to make the collection of data-owners behave like a single entity (without interacting with each other), so that they can communicate their collective data to the two servers in the form in which the underlying single data-owner scheme communicates it. Note that from this collective data, neither server should be able to link the records that are selected by a search query back to the individual data owners from whom it originates. In principle, this problem can be solved generically using secure multi-party computation techniques. However, for efficiency reasons, we develop a suite of techniques under the name of onion secret-sharing, that carefully combines secret-sharing and public-key encryption to achieve this.

Onion secret-sharing is a non-trivial generalization of the traditional mix-nets [18]. 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 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 (except for the senders whom they collude with). This is easily achieved as follows: each D_i sends $[\![M_i]\!]_{PK_\mathsf{S}}$ to A where PK_S is the public-key of S for a semantically secure public-key encryption scheme, and the notation $[\![M]\!]_{PK}$ denotes encryption of M using a public-key PK. A collects all such ciphertexts, sorts them lexicographically (or randomly permutes them), and forwards them to S; S decrypts them to obtain the multiset $\{M_1, \cdots, M_m\}$.

Now consider the following 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, the association between σ_i and ρ_i needs to be retained. Onion secret-sharing provides a solution to this problem, as follows:

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

- D_i 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

[6] $\mathcal{I}$ 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.

High Level Overview of query phase of $\mathrm{MC\text{-}OXT}$ in [40]: (See Appendix C for a longer discussion)

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, $\mathcal{L}_{\mathrm{MC-OXT-mod}}$:

- 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: $\mathrm{TIME}_{\mathsf{D}}^{\mathrm{init}} = O(\sum_{w\in\mathcal{K}} |DB(w)|)$; $\mathrm{TIME}_{\mathsf{A}}^{\mathrm{query}}, \mathrm{TIME}_{\mathsf{S}}^{\mathrm{query}} = O(|\mathrm{SP}|)$; $\mathrm{TIME}_{\mathsf{C}}^{\mathrm{query}} = O(1)$. For notation, see Table III.

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

- **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 AHE 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. $Enc(K, \mathrm{id})$ is replaced by $\langle\langle \mathrm{id}\rangle\rangle_{PK_{\mathsf{A}}}$.
- **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 $\{c_{\mathrm{id}}\}_{\mathrm{id}\in T}$ where $c_{\mathrm{id}} = \langle\langle \mathrm{id}\rangle\rangle_{PK_{\mathsf{A}}}$. For each id, it chooses a random value and adds the random value to the ciphertext. i.e. $\gamma_{\mathrm{id}} = \langle\langle r_{\mathrm{id}}\rangle\rangle_{PK_{\mathsf{A}}} + c_{\mathrm{id}}$ and $\delta_{\mathrm{id}} = \langle\langle -r_{\mathrm{id}}\rangle\rangle_{PK_{\mathsf{S}}}$. S sends $\{(\gamma_{\mathrm{id}}, \delta_{\mathrm{id}})\}_{\mathrm{id}\in T}$ to A.
  4) A decrypts using SK_{A} to reveal $\{(\mathrm{id}+r_{\mathrm{id}}, \delta_{\mathrm{id}})\}_{\mathrm{id}\in T}$. A encrypts using PK_{S} and calculates $\{\eta_{\mathrm{id}}\}_{\mathrm{id}\in T}$ where $\eta_{\mathrm{id}} = \langle\langle \mathrm{id}+r_{\mathrm{id}}\rangle\rangle_{PK_{\mathsf{S}}} + \delta_{\mathrm{id}}$. It permutes this set and sends it to S. (Note that: $\eta_{\mathrm{id}} = \langle\langle \mathrm{id}+r_{\mathrm{id}}\rangle\rangle_{PK_{\mathsf{S}}} + \langle\langle -r_{\mathrm{id}}\rangle\rangle_{PK_{\mathsf{S}}} = \langle\langle \mathrm{id}\rangle\rangle_{PK_{\mathsf{S}}}$)
  5) S decrypts using SK_{S} to reveal $\{\mathrm{id}\}_{\mathrm{id}\in T}$.

  We note that there is a need for permuting the plaintext id's from the ciphertexts to stay consistent with the exact leakage to S in MC-OXT. Otherwise, S learns statistics about each id that was decrypted due to the intersections with other keywords that it calculated.

Leakage, $\mathcal{L}_{\mathrm{MC-OXT-mod-adhom}}$:

- 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: $\mathrm{TIME}_{DO}^{\mathrm{init}} = O((\sum_{w\in\mathcal{K}} |DB(w)|)\mathrm{TIME}_{\mathrm{HomEnc}})$; $\mathrm{TIME}_{\mathsf{A}}^{\mathrm{query}}, \mathrm{TIME}_{\mathsf{S}}^{\mathrm{query}} = O(|\mathrm{SP}| + |T|(\mathrm{TIME}_{\mathrm{HomEnc}} + \mathrm{TIME}_{\mathrm{HomDec}}))$; $\mathrm{TIME}_{\mathsf{C}}^{\mathrm{query}} = O(1)$.

Modification by Removing Encryption:

- **Initialization Phase:**
  1) D proceeds exactly as in MC-OXT, it stores the plaintext in clear, i.e. $Enc(K, \mathrm{id})$ 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. $\{\mathrm{id}\}_{\mathrm{id}\in T}$

Leakage, $\mathcal{L}_{\mathrm{MC-OXT-mod-noenc}}$:

- 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: $\mathrm{TIME}_{\mathsf{D}}^{\mathrm{init}} = O(\sum_{w\in\mathcal{K}} |DB(w)|)$; $\mathrm{TIME}_{\mathsf{A}}^{\mathrm{query}}, \mathrm{TIME}_{\mathsf{S}}^{\mathrm{query}} = O(|\mathrm{SP}|)$; $\mathrm{TIME}_{\mathsf{C}}^{\mathrm{query}} = O(1)$.

Fig. 4: Security-Efficiency tradeoffs in modified MC-OXT in [40].

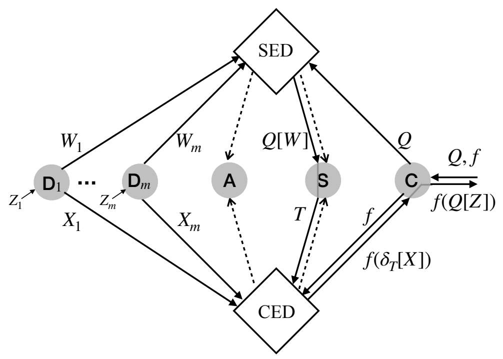


Fig. 5: FED protocol template. Each data-owner splits its input data Z_i as W_i and X_i, which it inputs to SED and CED, respectively. W denotes the combined data-set $\bigcup_i W_i$.

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 V-C2, we use a variant of the above scheme to let A tag entries in various lists with a pseudonym (multiple lists may get the same pseudonym), before the lists are unpacked and shuffled again, destroying the linkage to the lists, but retaining the tags. [7]

B. *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 — merge-map and map— collectively referred to as the functionality mmap, as described below.

Functionality Pair mmap = (merge-map, map)

- Functionality merge-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 $\hat{W} = \{(\hat{w}, \hat{\mathrm{id}}) | (w, \mathrm{id}) \in W, \hat{w} = M_{K_{\mathsf{map}}}(w)$, and $\hat{\mathrm{id}} = M_{K_{\mathsf{map}}}(\mathrm{id})\}$, where the mapping function M is to be specified when instantiating this template.[8] 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 $\texttt{SED-templ}^{s\mathsf{SED},\mathsf{mmap}}$ is shown in Figure 6. In this protocol, sSED is invoked with A playing the role of the data-owner as well. The invocation of merge-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 M.[9] In addition, leakages from merge-map, map and sSED (the latter on the merged dataset) will be included in the leakage of this protocol.

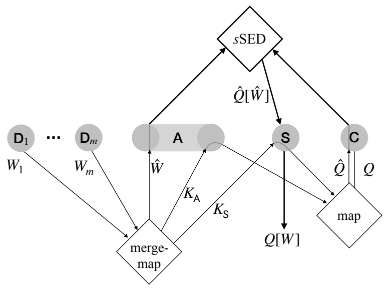


Fig. 6: SED protocol template using three functionalities sSED, merge-map and map. Although not shown, all three functionalities used may specify leakage to S and A. In accessing sSED, A plays the role of both the (single) data-owner and the auxiliary server.

C. *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 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 M, while the other relies

[7]For simplicity, here we consider the tagging to be arbitrary, whereas in Section V-C2, it is done based on equality checks. Here we allow A to add tags while it knows the link between data-items and D_i; in our application, this link is broken by an extra round of mixing.

[8]For notational simplicity, we have specified the mapping using a single function 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}.

[9]Such leakage could be avoided by relying on secure two-party computation of a certain function between A and S during initialization, but with high communication costs.

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 [40], 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 E.

1) Construction `mmap-OPRF`*:* The pair of protocols for the functionalities merge-map and map, collectively called `mmap-OPRF`, is shown in Figure 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}. 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 merge-map and map protocols. That is, for $w \subseteq \mathcal{K}$ we have $M_{K_{\mathsf{map}}}(w) = \{F_K(\tau) | \tau \in w\}$, 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 $\mathcal{K}_i = \{\tau | \exists \text{id s.t. } (\tau, \text{id}) \in \widetilde{W}_i\}$ 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, $\mathcal{L}^{\text{OPRF}}_{\text{mmap}}$ **:** 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 $\mathcal{L}^{\text{OPRF}}_{\text{mmap}}$ 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-templ` protocol that uses this functionality). $\hat{W}$ reveals an anonymized version of the keyword-identity incidence matrix of the merged data,[10]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, it learns the actual keyword labels of all the corresponding keywords held by the colluding data-owners.

2) 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 Figure 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. Please refer to Appendix D2 for a description of the protocol.

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. How this can be accomplished using the onion secret-sharing technique was already discussed in Section V-A.

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; the number of entries in individual lists for each keyword, and the total number of lists from each data owner are padded up; 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 Figure 8.

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 D3, we provide a mechanism to improve the efficiency of this construction, using a partition of the keyword space into bins.

Leakage, $\mathcal{L}^{\text{SFE}}_{\text{mmap}}$ **:** The leakage includes $\mathcal{L}^{\text{OPRF}}_{\text{mmap}}$ 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. In Appendix D4, we describe how the leakage may be further reduced.

[10]This is a 0-1 matrix with 1 in the position indexed by (τ, id) iff $(\tau, \text{id}) \in \bigcup_i \widetilde{W}_i$. 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.

D. *Security Analysis of* SED *protocols*

In this section, we provide a detailed security analysis of our SED protocols.

We focus on the instantiation of the SED protocol template (Section V-B) using the components developed in Section V-C1. The analysis of the full SED protocol breaks down into the analysis of the template protocol (Section V-B) 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 Figure 7. Instead of directly using the random oracle model, we assume an ideal OPRF functionality (which is then UC-securely realized in the random oracle model). 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 Figure 7 is easily seen to be a UC-secure realization of the corresponding functionality as it directly uses the OPRF *functionality*. As described in Section V-C1, we UC-securely realize this functionality against active corruption of the receiver, in the random oracle model, by using the OPRF protocol of [41].

Finally, we turn to the analysis of the SED template protocol (see Figure 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 IV-E, 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 Figure 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. 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 V-C2. As this analysis is not particularly enlightening, we omit the details and point the reader to the discussion on onion secret-sharing (Section V-A) for the intuition underlying the construction and the analysis.

E. CED *Protocols*

Upgrading an sCED protocol to a CED protocol is simpler than the sSED to SED transformation: Since there are no

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.

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 IV-D are modified in this manner.

Value Retrieval and Summation: Below we describe how the sCED scheme in Section IV-D 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 E.

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.
  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, $\mathcal{L}_{\mathtt{ValRet}}$ **:** 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.)

General Functions: The sCED scheme for general functions from Section IV-D (based on the CFE scheme of [58]) can be easily adapted to the multi data-owner setting. Since the original CFE scheme of [58] was already designed for multiple data-owners, but without any anonymization requirement on the inputs, the only modification needed in this case is to route the data from the data-owners to S anonymously (with the help of A). Below we present the changes required to the sCED scheme presented in Figure 3.

Protocol `CktEval`

- **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) in Figure 3 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 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,** $\mathcal{L}_{\mathtt{CktEval}}$ **:** On initialization, the ID-set $\mathcal{J} = \bigcup_j \mathcal{J}_j$ is leaked to S, and the sizes $\{|\mathcal{J}_j|\}_j$ 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.

VI. Putting it all Together

The various SED and CED protocols described so far can be assembled together into a complete FED system with different functionalities, by plugging them into the template protocol `FED-templ`. Similarly, sSED and sCED protocols can be plugged into the protocol `sFED-templ` to create a sFED system. For easy reference, we have summarized these modules in Table II. The resulting FED and sFED protocols inherit its functionality, efficiency and leakage characteristics from the modules used.

In Section VII, we illustrate how such combinations can be used to implement high-level tasks (in Genome Wide Association Studies), measuring their efficiency in different dimensions. Here, we briefly summarize the leakage characteristics of these high-level protocols. In all cases, the data-owners and clients obtain no leaked information[11] . Only the two servers S and A, who are assumed not to collude, obtain the leakage as specified in the description the templates `FED-templ` and `sFED-templ`. As an example, in our implementations of MAF and ChiSq from Section VII, the two servers learn – apart from the database schema and the description of the function being computed – the total number of records in the database. When a client makes a search query, A learns the

[11]We consider the leakage only from the messages received by the client. This does not account for leakage from side-channels like the time taken for answering a query, which does depend on the size of the number of records that get filtered by the query. The timing side-channel can be throttled by fixing the total wait time before the answer is returned to the client, and by regulating the rate at which queries are accepted to avoid loads that can possibly delay the answers.

query, and both servers learn the *pattern* of the search outcome (i.e., how many records are common with the outcomes of previous search queries). Neither server learns the result of the computation query (the MAF or ChiSq value).

When multiple data-owners are involved, the initialization phase leaks certain additional statistics to the two servers, depending on which instantiation of the protocol pair $(\mathsf{merge\text{-}map}, \mathsf{map})$ is used. In particular, during the initialization phase A can reconstruct an anonymized version of the merged database. We point out that an adversary who compromises A after the initialization phase does not obtain this additional leakage, because at the end of the initialization phase, A deletes this and retains only short keys for the query phase Nevertheless, in handling sensitive genomic data, further security is desirable. In Appendix D4, we describe how this leakage can be eliminated, at the expense of restricting the search queries to single fields in the database.

VII. Implementation and Experimental Results

In this section, we report details of our implementation and the experiments we performed to demonstrate the scalability and applicability of our protocols. In Subsection VII-A, we first show the feasibility and performance of our protocols on realisitically large-scale computations by running tasks representative of *Genome Wide Association Studies (GWAS)*. Next, in Subsection VII-B, we show how the experiments we perform validate the claims made about our protocols.

System/Experimentation details The whole system was implemented in C++11 with STL support. Various opensource cryptographic libraries (including OpenSSL, libPaillier, NTL, JustGarbled and Obliv-C) were used to implement various components in our protocol (more details in Appendix F). The MC-OXT protocol of Jarecki et al. [40] was reimplemented for use in our SED protocols. All experiments were performed on a Linux system with 32 GB RAM and Intel Core i7-6700 CPU running at frequency of 3.4GHz. The communication delays were estimated in a simulated local area network (simulated using linux tc command), with an average bandwidth of 620 MBps and ping time of 0.1 ms.

A. GWAS functionalities

The tasks we used for evaluation are representative of *Genome Wide Association Studies (GWAS)*, widely used to study associations between genetic variations and major diseases [9]. The specific functions we choose were adapted from the *iDash competition* [38], [45] 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, diseases, etc.), as well as genetic attributes. The genetic attributes of interest – called Single Nucleotide Polymorphisms (SNPs) – describe, for each of several loci (i.e., positions) in the genome, the "genotype" at that locus (typically, one of a small number of possibilities). In a typical query, the demographic and phenotypic attributes are used for filtering (e.g., South Asian males with type 2 diabetes [46], [12]), and statistics are computed over the genetic information. In our experiments, the following statistics were computed (more details in Appendix F).

- **Minor Allele Frequency (MAF)**: MAF for a locus of interest is defined as the real number $\frac{\min(n_A, 2N - n_A)}{2N}$ [45], where N is the total number of individuals filtered by a query Q, and n_A is the sum of their values in one field. This is an instance of a composite query of the form $f_0(f_1(Q[Z]), f_2(Q[Z])$, where f_1 computes $N = |Q[Z]|$, f_2 computes n_A, and f_0 computes the above formula (upto a finite precision). f_2 can be implemented using protocol `sSum` or `sAlt-Sum`, and the overall function f_0 using protocol `sComposite` (based on a garbled circuit for f_0).
- **Chi-square analysis (ChiSq)**: In this, two groups of individuals, of sizes N and N', are first selected using two search filters Q and Q' respectively. χ^2 statistic [45], [1] is then defined as $\frac{(N+N')(n_A N' - n'_A N)^2}{NN'(n_A + n'_A)((N+N') - (n_A + n'_A))}$ where n_A and n'_A are the sums of a certain field in the two groups. 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 (using protocols `sSum` or `sAlt-Sum`), and f_0 implements the above formula (up to finite precision) (using protocol `sComposite`).
- **Average Hamming Distance**: Hamming distance between 2 genome sequences, often used as a metric for genome comparison [81], [64], is defined $f_{x^*}(Q[Z]) = \frac{\sum_{x \in Q[Z]} \Delta(x, x^*)}{|Q[Z]|}$, where Δ stands for the hamming distance of two strings. Here, we consider the entire genotype vector x for each individual (rather than the genotype at a given locus). We used our protocol for general functions, `sCktEval`, to evaluate this.
- **Genome retrieval (GenomeRet)**: This is a retrieval task where we retrieve the genome data at some locus for the set of individuals selected by the search filter. We implement this using `sValRet` protocol.

Dataset and Queries: We conducted our experiments on synthetic data generated randomly. Our dataset sizes are typically between $10000 - 100000$ records and 50 SNPs, inspired by real-world applications [35], [78], [80]. For the Hamming distance functionality, we used $5000 - 25000$ records and 50 SNPs. For searching on these databases, we chose our queries so that the number of filtered records ranged between $600 - 3500$ for the Hamming Distance functionality and between $2000 - 12000$ for the other functionalities. All the reported values are averaged over the same 10 queries to account for experimental variation.

Metrics: Our results are reported over two metrics – the total time taken and the total communication size, across all entities. The reported time excludes the time taken to read/write to disk, in case the database is large. These metrics are reported separately for the initialization and query phases. Also, the costs incurred by the SED component of the protocols are shown, as this could be taken as a baseline cost that applies to the search component alone.

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.

B. Observations

a) Comparison to SSE: The costs of the search component in our single data-owner protocol is essentially the same as that of the underlying SSE protocols. In Figure 9, we show the break up of our sFED protocol costs between the search and compute components. As can be seen from the figure, on applications other than Av. Hamming, sFED has an overhead between 1.2x to 1.7x over sSED. For Av. Hamming, which computed a very large function using the protocol `sCktEval` for general protocols, our overhead is around 22x. Nevertheless, the computation remains feasible even in this case: for the parameters of Figure 9, we evaluate Hamming distance between strings of length around 125k in about 10 seconds.

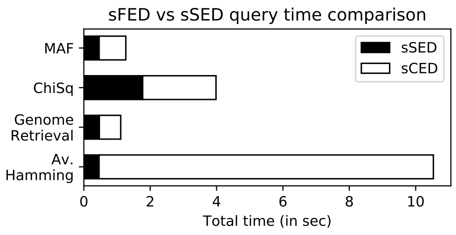


Fig. 9: sFED vs sSED comparison for different functionalities.

b) Linear in Data: In Figure 10, we plot the initialization and query times (summed up over all parties, with serial computation and LAN communication) for the various functions, each one using our different protocols. The initialization times are plotted against the total number of records in the database, while the query times are plotted against the number of records filtered out by the search query. As expected, all the times are linear in the number of records. [12]

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, even though only a small number of data-owners were used here. The additional slowdown in the FED-SFE setting exists because of the nesting of public key ciphertexts in this protocol. The overhead in the query phase is a more modest 5x factor. This overhead can be attributed to the single public key decryption operation performed in the query phase of FED.

The communication costs for the GWAS functionalities (Figure 11) follow a similar linear trend. As expected, the initialization costs are high (in the order of 1 GB for our largest benchmarks), whereas the query phase costs are a few MBs (except in the case of Av. Hamming distance, which has the servers carry out a large circuit evaluation using garbled circuits).

c) Efficiency of query phase: As expected from our complexity analysis, and as can be seen in Figure 10, the query time for our protocols is linearly proportional to the number of filtered records. A similar linear trend is observed for the communication costs (Figure 11). From our complexity analysis, we can observe that this is indeed the case for CED and sCED but not SED,sSED. The search functionality uses the protocols of MC-OXT of [40]. The protocols there have a complexity equal to the number of documents matching the first keyword and not the entire conjunction. Their work experiments on datasets such as the enron dataset to show that if the least frequent keyword is chosen as the first keyword, the query times are nearly linear in observation. Our 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.[13] This restriction helps us simplify the comparison of the FED protocols with their SED components. We provide some more observations for the interested reader in Appendix F.

d) Scalability with Number of Data-Owners: Our experiments confirm the ability of our protocols to scale to large number of data owners. Specifically, in Figure 12, we explore how well the initialization phase of our FED protocol scales with increase in number of data owners in the system, when each data owner provides only one record. This setting fits well with our motivation, where each user can be imagined as holding his/her genomic data and participating in the system to allow genetic researchers to query for statistical information later. The corresponding time in the plot includes the total time taken from when the data owners create their encrypted records to the end of the merge-map phase, assuming each of the owners is serviced serially by S. As can be seen from the plot, our OPRF based protocol scales to 100k data owners in around 500 seconds, while our SFE-based protocol scales to 2000 owners in a similar time.[14] Note that the latter scales quadratically, rather than linearly. We remark that in Appendix D3, we address this inefficiency, but our prototype implementation does not include these optimizations.

e) 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.

Interested readers can refer to Appendix F for further experiments, including an analysis of our `Alt-Sum` protocol.

VIII. Conclusion

We give a summary of the protocols proposed, their functionality and underlying security assumptions in Table II.

An influential body of research in SSE has established the possibility of searchably encrypting databases, with strong security and good performance guarantees [21], [11], [10], [63], [59], [23], [8]. In this work we introduced the notion of FED, which provides significantly richer functionality, with efficiency and security guarantees comparable to that of SSE. FED offers an alternative to tools like CryptDB, Monomi,

[12]Technically, the init phase of our FED protocols involve a sorting step, which requires more time than linear in the number of records. However, this cost is dominated by the use of heavy public-key cryptography and which grows as linear in the number of records.

[13]Note that on an arbitrary dataset, our CED protocols will still run in time linear in the filtered dataset size, but SED query time is linear in frequency of the query's least frequent keyword – which may not equal the size of the filtered dataset in general.

[14]In our experiments data owners were serviced serially by S. Though our prototype implementation does not exploit parallelism, we estimate a drop of 100-200 sec for our largest benchmarks if this were to be done parallely.

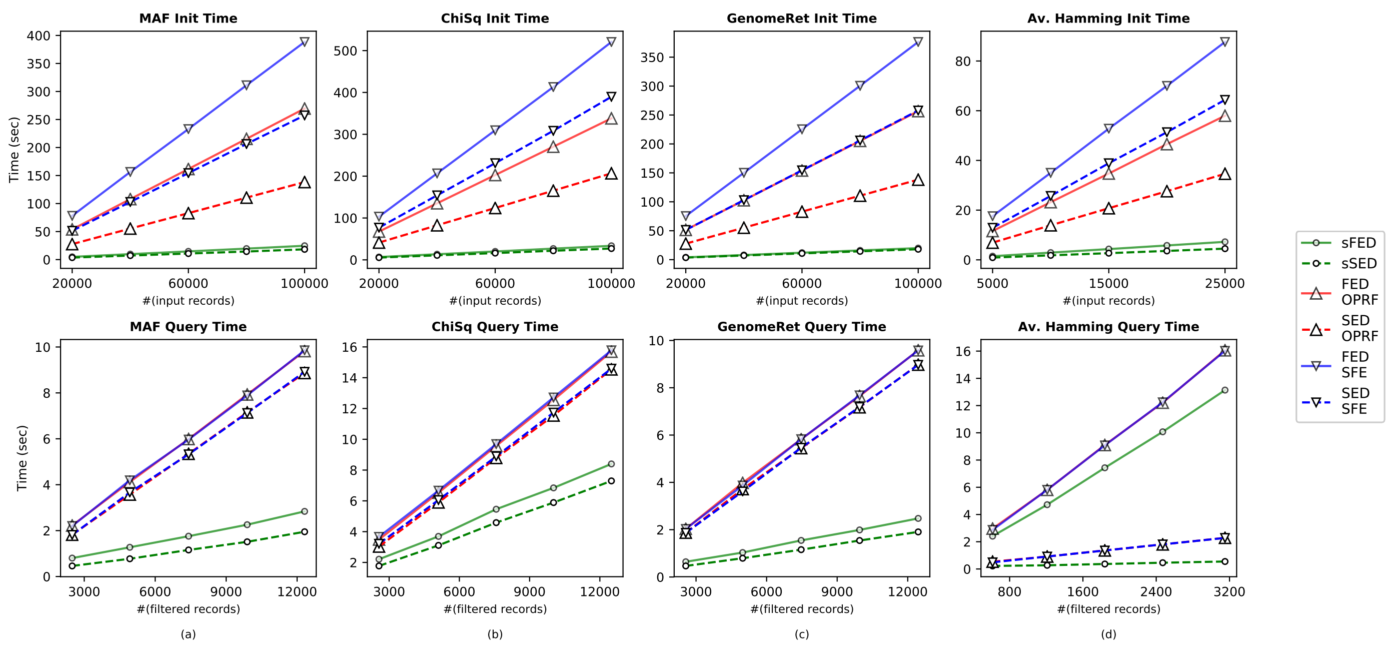


Fig. 10: Total time comparison in the initialization and query phases for single and multi data owner setting (contrasting two SED protocols, OPRF - Section V-C1, SFE - Section V-C2) for different applications: MAF, ChiSq, GenomeRet and Av.Hamming.

TABLE II: Summary of protocols proposed, their functionality and underlying security assumptions

Protocol Family	Instantiation	Reference	Functionality	Security Assumptions
sSED	`MD-MC-OXT`	Section IV-E	Search	Non-collusion (A and S), Decision Diffie-Hellman, Random Oracle Model, Secure PRFs
sCED	`sValRet`	Section IV-D	Value Retrieval	Non-collusion (A and S), Secure PRFs
sCED	`sSum`	Section IV-D	Summation	Non-collusion (A and S), Secure PRFs
sCED	`sComposite`	Section IV-D	$f_0(f_1(Q_1[Z]), \cdots, f_d(Q_d[Z]))$	Non-collusion (A and S), Oblivious Transfer
sCED	`sCktEval`	Section IV-D	Poly-time computable functions	Non-collusion (A and S), Secure PRFs
sCED	`sAlt-Sum`	Appendix B	Summation	Non-collusion (A and S), Secure PRFs, Public-Key Additive Homomorphic Encryption
SED	`mmap-OPRF, MD-MC-OXT`	Section V-C1	Multiple D's + Search	Non-collusion (A and S), Decision Diffie-Hellman, Random Oracle Model, Secure PRFs, One More Diffie-Hellman, Public Key Encryption
SED	`mmap-SFE, MD-MC-OXT`	Section V-C2	Multiple D's + Search	Non-collusion (A and S), Decision Diffie-Hellman, Random Oracle Model*, Secure PRFs, Public Key Encryption
CED	`ValRet`	Section V-E	Multiple D's + Value Retrieval	Non-collusion (A and S), Secure PRFs, Public Key Encryption
CED	`Sum`	Section V-E	Multiple D's + Summation	Non-collusion (A and S), Secure PRFs, Public Key Encryption
CED	`CktEval`	Section V-E	Multiple D's + Poly-time computable functions	Non-collusion (A and S), Secure PRFs, Public Key Encryption
CED	`Alt-Sum`	Appendix B	Multiple D's+Summation	Non-collusion (A and S), Secure PRFs, Public-Key Additive Homomorphic Encryption, Public-Key Encryption

*Random Oracle Assumption is not needed for `mmap-SFE`, but is needed for `MC-OXT-mod`

Note that using our template protocols, Section IV and Section V, we can compose any (s)CED and (s)SED protocol to give (s)FED.

Seabed and Arx, which provide similar functionality but with much weaker security guarantees.

Towards realizing this new notion, we developed new provably secure cryptographic protocols that carefully adapted existing protocols for SSE and CFE, combining them with novel techniques like onion secret-sharing. We implemented and tested our protocols on tasks from GWAS, which show that our candidate constructions are quite practical: databases with 100,000 records can be queried in a few seconds with a few MB of communication, after an initialization phase that takes a few hundred of seconds even if the data is provided by 100,000 separate data-owners.

Acknowledgements

We would like to thank Rohan Kadekodi for helpful discussion on the experimentation section and anonymous reviewers

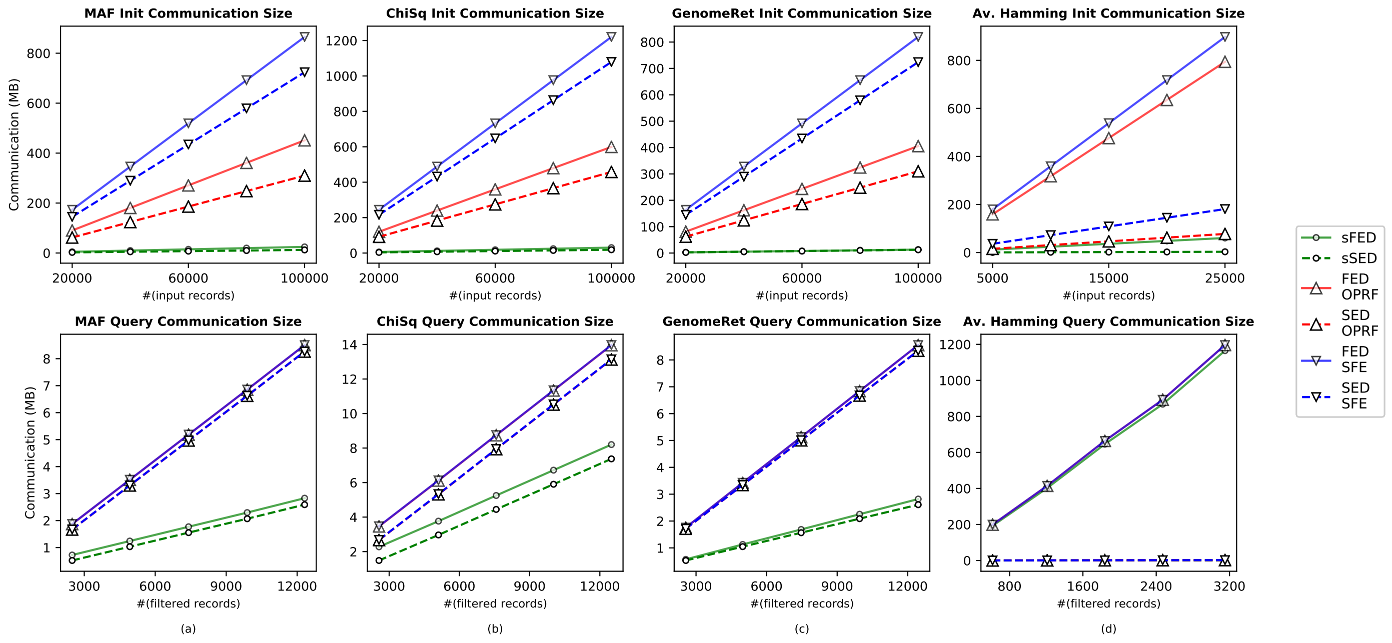


Fig. 11: Total communication across all entities comparison in the initialization and query phases for single data owner and multi data owner setting (contrasting two SED protocols, OPRF - Section V-C1, SFE - Section V-C2) for different applications.

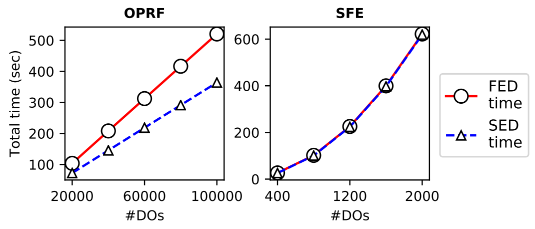


Fig. 12: Scaling of initialization time of FED with increasing number of data-owners (each holding one record)

for their comments.

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

References

[1] J. Baker, “Comparing two populations,” http://grows.ups.edu/curriculum/Comparing%20Two%20Population1.htm.

[2] M. Bellare, V. T. Hoang, and P. Rogaway, “Foundations of Garbled Circuits,” in *CCS*, 2012.

[3] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, “Efficient garbling from a fixed-key blockcipher,” Cryptology ePrint Archive, Report 2013/426, 2013.

[4] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, “Multi-prover interactive proofs: How to remove intractability assumptions,” in *STOC*, 1988.

[5] J. Bethencourt, “Paillier library,” 2006, http://acsc.cs.utexas.edu/libpaillier/.

[6] G. R. Blakley *et al.*, “Safeguarding cryptographic keys,” in *Proceedings of the national computer conference*, vol. 48, no. 313, 1979.

[7] D. Boneh, A. Sahai, and B. Waters, “Functional encryption: Definitions and challenges,” in *TCC*, 2011, pp. 253–273.

[8] R. Bost, “Σoφoς: Forward secure searchable encryption,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1143–1154.

[9] W. S. Bush and J. H. Moore, “Genome-wide association studies,” *PLoS computational biology*, vol. 8, no. 12, p. e1002822, 2012.

[10] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, “Dynamic searchable encryption in very-large databases: Data structures and implementation,” in *21st Annual Network and Distributed System Security Symposium, NDSS*, 2014.

[11] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, “Highly-scalable searchable symmetric encryption with support for boolean queries,” in *CRYPTO*, 2013.

[12] J. C. Chambers, J. Abbott, W. Zhang, E. Turro, W. R. Scott, S.-T. Tan, U. Afzal, S. Afaq, M. Loh, B. Lehne *et al.*, “The south asian genome,” *PLoS One*, vol. 9, no. 8, p. e102645, 2014.

[13] Y.-C. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Applied Cryptography and Network Security*. Springer, 2005, pp. 442–455.

[14] M. Chase and S. Kamara, “Structured encryption and controlled disclosure,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 577–594.

[15] ——, "Structured encryption and controlled disclosure," in *ASIACRYPT*, 2010.

[16] M. Chase and E. Shen, "Substring-searchable symmetric encryption," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 263–281, 2015.

[17] ——, "Substring-searchable symmetric encryption," *PoPETs*, vol. 2015, 2015.

[18] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981.

[19] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *FOCS*, 1995.

[20] G. M. Clarke, C. A. Anderson, F. H. Pettersson, L. R. Cardon, A. P. Morris, and K. T. Zondervan, "Basic statistical analysis in genetic case-control studies," *Nature protocols*, vol. 6, no. 2, p. 121, 2011.

[21] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM CCS*, 2006, pp. 79–88.

[22] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *ACM SIGMOD*, 2016.

[23] W. A. Drazen, E. Ekwedike, and R. Gennaro, "Highly scalable verifiable encrypted search," in *2015 IEEE Conference on Communications and Network Security, CNS*, 2015, pp. 497–505.

[24] D. Evans, J. Katz, and Y. Huang, "Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution," *2014 IEEE Symposium on Security and Privacy*, vol. 0, pp. 272–284, 2012.

[25] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin, "Malicious-client security in blind seer: a scalable private dbms," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 395–410.

[26] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–178.

[27] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[28] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987.

[29] O. Goldreich, *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[30] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou, "Multi-input functional encryption," in *EUROCRYPT*, 2014, pp. 578–602.

[31] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *International Conference on Applied Cryptography and Network Security*. Springer, 2004, pp. 31–45.

[32] P. Grubbs, T. Ristenpart, and V. Shmatikov, "Why your encrypted database is not secure," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. ACM, 2017, pp. 162–168.

[33] A. Hamlin, A. Shelat, M. Weiss, and D. Wichs, "Multi-key searchable encryption, revisited," in *IACR International Workshop on Public Key Cryptography*. Springer, 2018, pp. 95–124.

[34] "Havasupai tribe and the lawsuit settlement aftermath," http://genetics.ncai.org/case-study/havasupai-Tribe.cfm.

[35] M. Hirokawa, H. Morita, T. Tajima, A. Takahashi, K. Ashikawa, F. Miya, D. Shigemizu, K. Ozaki, Y. Sakata, D. Nakatani *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*, vol. 23, no. 3, p. 374, 2015.

[36] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11, 2011, pp. 35–35.

[37] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *CRYPTO*, 2013.

[38] "IDASH PRIVACY & SECURITY WORKSHOP 2015," http://www.humangenomeprivacy.org/2015/competition-tasks.html, 2015.

[39] "Indian tribe wins fight to limit research of its dna," https://www.nytimes.com/2010/04/22/us/22dna.html?pagewanted=all&_r=1&.

[40] S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *ACM CCS*, 2013.

[41] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "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)*. IEEE, 2016, pp. 276–291.

[42] S. Kamara and T. Moataz, "Sql on structurally-encrypted databases," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 149–180.

[43] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 965–976.

[44] R. Kerbaj and J. Ungoed-Thomas, http://www.thesundaytimes.co.uk/sto/news/uk_news/National/article1258476.ece?CMP=OTH-gnws-standard-2013_05_11.

[45] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," *BMC medical informatics and decision making*, vol. 15, no. 5, p. S3, 2015.

[46] J. S. Kooner, D. Saleheen, X. Sim, J. Sehmi, W. Zhang, P. Frossard, L. F. Been, K.-S. Chia, A. S. Dimas, N. Hassanali *et al.*, "Genome-wide association study in people of south asian ancestry identifies six novel susceptibility loci for type 2 diabetes," *Nature genetics*, vol. 43, no. 10, p. 984, 2011.

[47] B. Kreuter, A. Shelat, B. Mood, and K. R. B. Butler, "PCF: A portable circuit format for scalable two-party secure computation," in *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, 2013, pp. 321–336.

[48] B. Kreuter, A. Shelat, and C. Shen, "Billion-gate secure computation with malicious adversaries," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, 2012, pp. 285–300.

[49] K. Kurosawa and Y. Ohtaki, "Uc-secure searchable symmetric encryption," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 285–298.

[50] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast range query processing with strong privacy protection for cloud computing," *Proc. VLDB Endow.*, vol. 7, no. 14, 2014.

[51] Y. Lindell, "Fast cut-and-choose based protocols for malicious and covert adversaries," in *CRYPTO*, 2013.

[52] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *STOC*, 2012, pp. 1219–1234.

[53] S.-K. Low, A. Takahashi, T. Mushiroda, and M. Kubo, "Genome-wide association study: a useful tool to identify common genetic variants associated with drug toxicity and efficacy in cancer pharmacogenomics," 2014.

[54] S. Lu and R. Ostrovsky, "Distributed oblivious ram for secure two-party computation," in *Theory of Cryptography Conference*. Springer, 2013, pp. 377–396.

[55] J. MacArthur, E. Bowler, M. Cerezo, L. Gil, P. Hall, E. Hastings, H. Junkins, A. McMahon, A. Milano, J. Morales *et al.*, "The new nhgri-ebi catalog of published genome-wide association studies (gwas catalog)," *Nucleic acids research*, vol. 45, no. D1, pp. D896–D901, 2016.

[56] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay — a secure two-party computation system," in *In USENIX Security Symposium*, 2004, pp. 287–302.

[57] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, 2004.

[58] M. Naveed, S. Agrawal, M. Prabhakaran, X. Wang, E. Ayday, J.-P. Hubaux, and C. Gunter, "Controlled functional encryption," in *CCS*, 2014.

[59] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 639–654.

[60] I. M. official statement, https://www.ipsos-mori.com/newsevents/latestnews/1390/Ipsos-MORI-response-to-the-Sunday-Times.aspx.

[61] C. Orencik, A. Selcuk, E. Savas, and M. Kantarcioglu, "Multi-keyword search over encrypted data with scoring and search pattern obfuscation,"

International Journal of Information Security, vol. 15, no. 3, pp. 251–269, 2016.

[62] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with seabed." in *OSDI*, 2016, pp. 587–602.

[63] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin, "Blind seer: A scalable private dbms," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 359–374.

[64] C. D. Pilcher, J. K. Wong, and S. K. Pillai, "Inferring hiv transmission dynamics from phylogenetic sequence relationships," *PLoS medicine*, vol. 5, no. 3, p. e69, 2008.

[65] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system." *IACR Cryptology ePrint Archive*, vol. 2016, p. 591, 2016.

[66] G. S. Poh, M. S. Mohamad, and J.-J. Chin, "Searchable symmetric encryption over multiple servers," *Cryptography and Communications*, vol. 10, no. 1, pp. 139–158, 2018.

[67] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 85–100.

[68] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, and H. Balakrishnan, "Building web applications on top of encrypted data using mylar," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 157–172.

[69] R. A. Popa and N. Zeldovich, "Multi-key searchable encryption," Cryptology ePrint Archive, Report 2013/508, 2013, https://eprint.iacr.org/2013/508.

[70] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE J.Sel. A. Commun.*, vol. 16, no. 4, pp. 482–494, Sep. 2006.

[71] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT*, 2005, pp. 457–473.

[72] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[73] E. Shi, J. Bethencourt, T. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 2007, pp. 350–364.

[74] V. Shoup, "NTL: A library for doing number theory," https://www.shoup.net/ntl/.

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

[76] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proceedings of the 39th international conference on Very Large Data Bases*, ser. PVLDB'13. VLDB Endowment, 2013, pp. 289–300. [Online]. Available: http://dl.acm.org/citation.cfm?id=2488335.2488336

[77] US Government, http://www.census.gov/ces/dataproducts/index.html.

[78] P. van der Harst and N. Verweij, "The identification of 64 novel genetic loci provides an expanded view on the genetic architecture of coronary artery disease," *Circulation research*, pp. CIRCRESAHA–117, 2017.

[79] C. Van Rompay, R. Molva, and M. Önen, "Secure and scalable multi-user searchable encryption." *IACR Cryptology ePrint Archive*, vol. 2018, p. 90, 2018.

[80] B. F. Voight, L. J. Scott, V. Steinthorsdottir, A. P. Morris, C. Dina, R. P. Welch, E. Zeggini, C. Huth, Y. S. Aulchenko, G. Thorleifsson *et al.*, "Twelve type 2 diabetes susceptibility loci identified through large-scale association analysis," *Nature genetics*, vol. 42, no. 7, p. 579, 2010.

[81] C. Wang, W.-H. Kao, and C. K. Hsiao, "Using hamming distance as information for snp-sets clustering and testing in disease association studies," *PloS one*, vol. 10, no. 8, p. e0135918, 2015.

[82] A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986.

[83] S. Zahur and D. Evans, "Circuit structures for improving efficiency of security and privacy tools." in *IEEE Symposium on Security and Privacy*, 2013.

[84] ——, "Obliv-c: A language for extensible data-oblivious computation." *IACR Cryptology ePrint Archive*, vol. 2015, 2015.

[85] A. Zelin, https://www.mrs.org.uk/pdf/Andrew_Zelin_presentation.pdf.

Appendix

A. Notation

In Table III we summarize the notation and symbols used across the paper.

B. Alternate Protocol for Summation

Our protocol for summation in Section IV-D 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. $\text{TIME}^{\text{init}}_{\mathsf{D}} = O(|X|\text{TIME}_{\text{HomEnc}})$; $\text{TIME}^{\text{query}}_{\mathsf{A}} = O(\text{TIME}_{\text{HomDec}})$; $\text{TIME}^{\text{query}}_{\mathsf{S}} = O(|T| + \text{TIME}_{\text{HomEnc}})$; $\text{TIME}^{\text{query}}_{\mathsf{C}} = O(1)$.

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

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.

C. Brief Description of MC-OXT

In this section, we give a brief overview of the multi-client SSE scheme (MC-OXT) of [40]. 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 [40] 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 $\text{id} \in \mathsf{DB}$ as $\mathsf{xind} = F_p(K_I, \text{id})$. 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

TABLE III: 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	$\text{TIME}^{K}_{\text{OPRF}}$, $\text{TIME}_{\text{OPRF}}$	Time taken in the 2-party OPRF protocol by party with and without a key respectively

containing the word w_1 (our s-word) using a list T_{w_1}. Then we can check $\forall$ id $\in T_{w_1}$ if id contains the keyword w_2 by computing if $g^{F_p(K_X, w_2) \cdot F_p(K_I, \text{id})} \in \mathsf{XSet}$. 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 $e = \mathsf{Enc}(K_e, \text{id})$. 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 [40] 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 $\text{DB}(w)$ 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, $\Sigma_{w \in \mathcal{K}} |\text{DB}(w)|$. 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 $\text{RP[i]} = \text{DB}(s[i]) \cap \text{DB}(x[i])$.
- The number of documents matching the first keyword in the query, denoted by Size Pattern(SP). Formally, a vector of size $|Q|$ with $\text{SP} = |\text{DB}(s[i])|$.
- The number of queries which have equal first terms (s-words).
- The conditional intersection pattern(IP), which is formally a $|Q|$ by $|Q|$ table defined by $\text{IP}[i, j] = \text{DB}(s[i]) \cap \text{DB}(s[j])$, if ($i \neq j$ and $x[i] = x[j]$), otherwise ϕ.

Additionaly, we leak Q to D and SP to C. [40] describes OSPIR-OXT, a protocol which reduces leakage of Q to D, using $OPRF$ evaluations and policy access control.

Complexity of the protocol is D (Setup) - $O(\sum_{w \in \mathcal{K}} |DB(w)|)$; D, C and S (Query) - $O(|\text{SP}|)$. Please refer to [40] for a complete formal analysis.

D. SED *protocols: Additional Details*

1) Actively secure OPRF: In this section, we describe the UC-secure OPRF protocol of [41] 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 [41]) here, but refer the the reader to Theorem 1 of [41] for further details.

(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 $s \in [Q+1]$.

2) Protocol for computing equality checks: We use securely pre-computed correlations for a secure evaluation of an equality check between two parties. The strings to check for equality will be mapped using a collision resistant hash to, say 64 bits, and interpreted as elements in a field, say $\mathbb{F} = GF(2^{64})$. 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 $a = v$.

3) 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 Figure 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.)

4) Reducing SED *Leakage Further:* The template used in Section V-C 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 the leakage description in Section V-C1. 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 $\hat{W}$.

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 V-C1 and Section V-C2, 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 $(\tau, \text{id}) \in \widetilde{W}_i$).

E. Time Complexity

Here we summarize the time complexity of our various protocols.

sCED **Protocols of Section IV-D**

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 `sCktEval`: Suppose the circuit for f is of size s, and its output has m bits. Then,
   - $\text{TIME}_{\mathsf{D}}^{\text{init}} = O(|X|)$;
   - $\text{TIME}_{\mathsf{A}}^{\text{query}}, \text{TIME}_{\mathsf{S}}^{\text{query}} = O(|T|(s))$;
   - $\text{TIME}_{\mathsf{C}}^{\text{query}} = O(m)$.
4) 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 V-C**

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 V-E**

1) `ValRet`:
   - $\text{TIME}^{\text{init}}_{\text{D}_i} = O(|X_i| + |X_i|\text{TIME}_{\text{Enc}})$;
   - $\text{TIME}^{\text{init}}_{\text{S}} = O(|X| + |X|\log|X| + |X|\text{TIME}_{\text{Dec}})$;
   - $\text{TIME}^{\text{init}}_{\text{A}} = O(|X| + |X|\text{TIME}_{\text{Dec}})$;
   - $\text{TIME}^{\text{query}}_{\text{A}}, \text{TIME}^{\text{query}}_{\text{S}} = O(|T|)$; $\text{TIME}^{\text{query}}_{\text{C}} = O(1)$.
2) `CktEval`: Suppose the circuit for f is of size s, and its output has m bits. Then,
   - $\text{TIME}^{\text{init}}_{\text{D}_i} = O(|X_i|\text{TIME}_{\text{Enc}})$;
   - $\text{TIME}^{\text{init}}_{\text{A}} = O(|X|\log|X|)$;
   - $\text{TIME}^{\text{init}}_{\text{S}} = O(|X|\text{TIME}_{\text{Dec}})$.
   - $\text{TIME}^{\text{query}}_{\text{A}} = O(|T|(s) + |T|\text{TIME}_{\text{Dec}})$;
   - $\text{TIME}^{\text{query}}_{\text{S}} = O(|T|(s) + |T|\text{TIME}_{\text{Enc}})$;
   - $\text{TIME}^{\text{query}}_{\text{C}} = O(m)$.

F. 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 [40], we used NIST-224p elliptic curves.
- MC-OXT scheme of [40] was reimplemented and ensured that the performance is comparable to that reported in [40].
- Additive Homomorphic scheme was implemented using the Paillier Cryptosystem using the Libpaillier library [5].
- JustGarble [3] was used to implement the CED scheme for general function evaluation.
- Two-party computation for CED for composite queries was implemented using Obliv-C [84]. This provided an implementation of Oblivious Transfer and Yao's Garbled Circuit.[15]
- NTL (Number Theory Library) [74] version 11.3.0 was used to implement the field operations in our construction using secure equality checks.

GWAS Functionalities Here we elaborate some more details on the GWAS functionalities for the interested reader.

- 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 [45] 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 [20]. 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 [45], [1] 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).

[15]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.

Experiments

We elaborate on some more details of our experimentation below.

- MAF Computation on filtered inputs of size 12000 took close to 11 seconds for FED and close to 4 sec for sFED (See Figure 10(a)). This competes with GenomeRet.
- Chi-square Analysis The Chi-square formula was represented as a garbled circuit with 38000 gates. This required 0.23 sec for computation using the Obliv-C framework [84]. Note that this computation stayed invariant of the number of records filtered. Computation on filtered inputs of size 12000 took close to 24 sec for FED and close to 14 sec for sFED. Time taken was more than twice when compared to MAF or Genome Retrieval as there were two summation queries computed in the inner protocol and an additional polynomial computed in the outer protocol. Please see Figure 10(b) for details.
- Genome Retrieval As seen in Figure 10(c), sSED is the heavier part of the computation in sFED. Computation on filtered inputs of size 12000 took close to 11 and 3.7 sec for FED and sFED.
- Average Hamming distance In this case, the computation is heavy in the sCED stage and all trends observed due to the changes in sSED are hidden, as seen in Figure 10(d). On a population of size 25000, being filtered to 3100 individuals, the circuit being computed had approximately 80 million gates and 60 million input bits. Hence, the two functionalities sFED and FED (both protocols i.e. OPRF and SFE) perform similarly. Computation on filtered inputs of size 3100 took about 30 sec.
- Comparing the two summation protocols Our two summation protocols are compared in Figure 13. During initialization, while summation based on additive secret sharing took about 6 and 0.6 sec in the FED and sFED functionalities respectively, the alternate summation protocol took correspondingly 207 and 202 sec for the same dataset. Comparing FED and sFED, we note that during initialization the computation and hence performance are same. During the query phase, the two functionalities perform equally well.

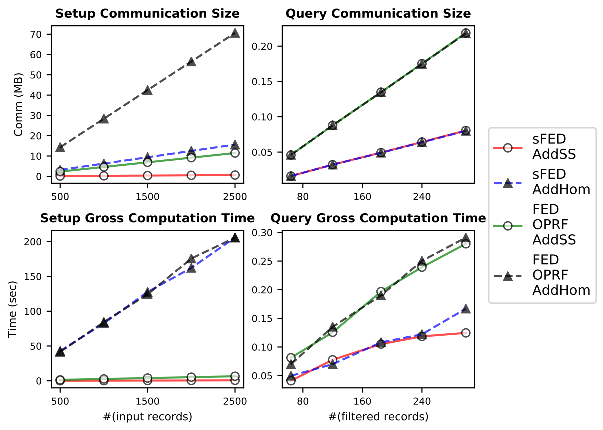


Fig. 13: Two CED protocols are contrasted here, one using the additive secret-sharing based summation (AddSS) and the other using the additive homomorphic encryption based summation (AddHom).