Secure Data Retrieval on the Cloud

Homomorphic Encryption meets Coresets

Adi Akavia[1], Dan Feldman[2] and Hayim Shaul[3]

[1] Haifa University, adi.akavia@gmail.com
[2] Haifa University, dan.feldman@gmail.com
[3] Haifa University hayim.shaul@gmail.com

Abstract. *Secure Report* is the problem of retrieving from a database table (e.g. on the cloud) all records matching specified attributes, as in SQL SELECT queries, but where the query and possibly the database are encrypted. Here, only the client has the secret key, but still the server (e.g. cloud owner) can compute and return the encrypted result. Secure report is theoretically possible with Fully Homomorphic Encryption (FHE). However, the current state-of-the-art solutions are realized by a polynomial of degree that is at least linear in the number m of records, which is too slow in practice even for very small databases. Nevertheless, in this work we present the first algorithm for secure report that is realized by a polynomial of degree polynomial in $\log m$, as well as the first implementation of secure (FHE) report. This is by suggesting a novel paradigm that forges a link between cryptography and modern data summarization techniques known as core-sets, and sketches in particular. The key idea is to compute only a core-set of the desired report. Since the core-set is small, the client can quickly decode the desired report that the server computes after decrypting its core-set. We implemented our main reporting system including all its sub-routines in an open source library. This is the first implemented system that can answer such database queries under the strong secure notion of FHE. As our analysis promises, the experimental results show that we can run secure report queries on billions records compared to few thousands in previous FHE papers. We hope that our results and open code would lead to the first FHE database engine in the near future.

Keywords: FHE · Search

1 Introduction

Storage and computation have become a commodity with many organizations and individuals (client) outsourcing storage and computation to large third-party systems often called "the cloud" (server). Usually this requires the client to reveal its private records to the server so that the server would be able to run the computations for the client. With e-mail, medical, financial and other personal information transferring to the cloud, it is paramount to guarantee *privacy* on top of data availability while keeping the correctness of the computations.

Fully Homomorphic Encryption (FHE) [56, 33, 34] is an encryption scheme that facilitates secure outsourcing of computation due to its special property of enabling computing on encrypted data; see a survey in [42]. Specifically, FHE allows computing any algorithm on encrypted input (ciphertexts), with no decryption or access to the secret key that would compromise secrecy, yet succeeding in returning the encryption of the desired outcome.

Secure outsourcing of computation using FHE is conceptually simple: the client sends the ciphertext $[\![x]\!]$ encrypting the input x and receives the ciphertext $[\![y]\!]$ encrypting the

output $y = f(x)$, where the computation is done on the server's side requiring no further interaction with the client. This gives a *single round* protocol and with *low communication*; specifically the communication complexity is proportional only to the sizes of the input and output ciphertexts (in contrast to communication proportional to the running time of computing $f()$ when using prior secure multi-party computation (MPC) techniques [62, 37]). The semantic security of the underlying FHE encryption ensures that the server learns no new information on the plaintext input and output from seeing and processing the ciphertexts.

A main challenge for designing algorithms that run on data encrypted with fully (or leveled) homomorphic encryption (FHE) is to present their computation as a *low degree polynomial* $f()$, so that on inputs x the algorithm's output is $f(x)$ (see examples in [52, 39, 49, 64, 17, 51, 30]). Otherwise, a naive conversion resulting in a high degree polynomial $f()$ would typically be highly impractical for the current state-of-the-art FHE implementations, where running time is rapidly growing with degree and the multiplicative depth of the corresponding circuit.

1.1 Secure Report Problem Statement

Secure report using FHE is a variant of secure search that has been a leading example for FHE applications since Gentry's breakthrough result construction the first FHE candidate [33]. We consider the *secure report* problem of retrieving from an unsorted data $array = (x_1, \ldots, x_m)$ all pairs

$$I = \{ (i, x_i) \mid i \in \{1, \ldots, m\}, x_i = \ell \} \tag{1}$$

of items matching a given lookup value ℓ where both input and output are encrypted FHE. In this example, we consider the matching criteria being the equality test. More generally, we consider a criteria given by a function $\text{isMatch}(x_i, \ell)$ which determines whether x_i matches the query ℓ. See discussion below. We remark that in the context of secure retrieval from a database, we think of *array* as the database table (or a specific column), and ℓ the query.

Our complexity goals are different for the server and for the client. Since the server evaluates a polynomial using FHE operations, our goal is to minimize its degree. The client, on the other hand runs a Turing machine and our goal is to minimize its running time. In addition, we aim for a protocol with as little communication as possible.

In the context of secure outsourcing, we focus on protocols with *low communication complexity*, where the size of the communication is proportional to the size of the input (the query) and the output. Here we assume *array* is pre-loaded to the server and not a part of the input.

Definition 1 (Secure Report)**.** The server holds an unsorted array of encrypted values (previously uploaded to the server, and where the server has no access to the secret decryption key):

$$[\![array]\!] = ([\![x_1]\!], \ldots, [\![x_m]\!])$$

(here and throughout this work, $[\![msg]\!]$ denotes the ciphertext encrypting message msg; the encryption can be any fully, or leveled, homomorphic encryption (FHE) scheme, e.g. [8]). The client sends to the server an encrypted lookup value $[\![\ell]\!]$. The server returns to the client a set of encrypted indices, value pairs

$$[\![y]\!] = \{([\![i_1]\!], [\![x_{i_1}]\!]), ([\![i_2]\!], [\![x_{i_2}]\!]), \ldots\}$$

satisfying the condition:

$$\text{isMatch}(x_i, \ell) = 1, \text{ for } i = i_1, i_2, \ldots$$

for isMatch() a predicate specifying the search condition (see discussion below on using generic predicates). More generally, y may be a value (sketch) from which the client can compute the set $\{(i_1, x_{i_1}), \ldots\}$ (decode).

We say the client / server / protocol is efficient if the following holds:

- The *client is efficient* if its running time is polynomial in the time to compute $|\ell|$ encryptions and $|I| \log m$ decryptions of the underlying FHE (for $|\ell| = \log m + |x_i|$ the bit representation length of the input and $|I|$ is size of the output as defined in 1).
- The *server is efficient* if the polynomial $f([\![array]\!], [\![\ell]\!])$ it evaluates to obtain $[\![y]\!]$ is of degree polynomial in $\log m$ and the degree of isMatch(), and of size (i.e., the overall number of addition and multiplication operations for computing f) polynomial in n and the size of isMatch.
- The *protocol is efficient* if both client and server are efficient and the communication is of size proportional to the size of the input, $|\ell|$, and the output, $|I| \log m$.

We call the client / server / protocol *inefficient* otherwise.

Generic isMatch() predicate. The predicate isMatch() in Definition 1 is a generic predicate that can be instantiated to any desired functionality (with complexity affected accordingly, see Theorem 1). Moreover, a concise specification of isMatch() can typically be used, e.g., by the client providing the function's name. For example, most generally, isMatch() can be a *universal circuit* and ℓ a full specification of the predicate defining the matching values. Alternatively, giving a more concrete instantiation, we can extend the search query to provide the name for a particular isMatch() circuit to be used, chosen from a commonly known set of options (for example, equality operator, conjunction/disjunction query, range query, similarity condition, and so forth). Even more concretely, we can fix a particular predicate in advance, say, the equality condition $\text{isMatch}(x_i, \ell) = 1$ if-and-only-if $x_i = \ell$. Looking ahead, our experiments are for the latter case; nonetheless, our results are general and apply to any generic isMatch() condition (see Theorem 1).

A toolbox of concrete instantiations of isMatch predicate on FHE encrypted data has been provided by prior works [63, 18, 50, 16, 47, 59, 48], including for example efficient implementations for computing Hamming and edit distances. These works focus on efficiently computing the isMatch predicate given the two values x_i and ℓ. This is then employed to either give a YES/NO answer on whether a match exists in the data array (but without returning the matching record or its index), or returning the length n indicator vector $\chi = (\text{isMatch}(x_1, \ell), \ldots, \text{isMatch}(x_m, \ell))$ indicating for each record x_i whether or not it is a match to the lookup value ℓ. In this work we focus on the complementary problem of retrieving the set $\{i_1, i_2, \ldots\}$, while using as a black-box the provided isMatch criterion.

Threat Model. The functionality that our protocols implement has client's input $(array, \ell)$; client's output the retrieved pairs $\{(i_1, x_1), \ldots\}$; and the server has no input or output beyond the shared parameters, including most notably, the number of data records, the sizes of records and lookup values and the size of the output. In Section 3.1.2 we discuss how the size of the output can be hidden from the server if we allow inefficient communication.

The adversaries we address are computationally-bounded semi-honest adversaries controlling the server. Namely, the adversary follows the protocol, but may try to learn additional information. We point out that there is no need to consider adversaries controlling the client because the server has no input/output.

Our security requirement is that the adversary learns no information from participating in the protocol beyond what is explicitly leaked in the shared parameters. In particular, if

the client issues the protocols with one of two adversarially-chosen equal size lookup values (similarly, data arrays), the adversary controlling the server cannot distinguish between them. Looking ahead, security follows immediately from the semantic security of the FHE scheme.

Applications of secure report on FHE encrypted data are abundant in the context of secure outsourcing of computation to an untrusted party ("the cloud"/ the server), examples including:

- Secure retrieval from a database
- Secure search in images corpus' tags
- Secure search in text documents
- Secure search in genomic data

In all applications, the lookup value and data (database, images/tags, text documents, genomic data) are encrypted by the client prior to being sent to the server, to ensure their secrecy.

The most relevant applications arise in settings where data pre-sorting or pre-indexing is infeasible, and linear scan of the data is used regardless of security.[1] These are particularly appealing scenarios due to known lower bounds necessitating a linear scan for securely searching on FHE encrypted data. Such applications arise for example in settings with:

- A-priori unknown matching criterion, as in ad-hoc SQL queries and our generic isMatch predicate, where indexing is impossible;
- Versatile matching criteria requiring indexes of size exponential in the number of attributes, as in range queries on high dimensional data;
- Streaming data with each element discarded by the client immediately after being encrypted and uploaded to the server, making pre-sorting the entire clear-text data impossible;
- Low capacity client that is too weak to store/sort the clear-text data, as in Internet-of-Things (IoT) devices;
- Fragmented data uploaded to the server from multiple distinct clients (e.g., agents/users/devices) with no single entity that holds and can pre-sort the entire clear-text data.

Our main motivation in this paper is to answer affirmatively the following question: **Is there an *efficient* secure report protocol?**

1.2 Related Works

Secure search has been extensively studied employing a variety of cryptographic tools leading to solutions with versatile properties.

Most relevant scenario: Secure search on FHE encrypted data. The scenario most relevant to our settings is secure search on FHE encrypted data [34], when focusing on the single server settings and on protocols with a single-round communication of bandwidth growing only with the size of the encrypted input and output. We note that a linear scan of the data on the server's side is necessary for those settings.

Private Information Retrieval (PIR) [20] constructions for these settings appeared early on [33, 9, 29]. PIR allows a client to retrieve data from a server while hiding both the

[1] Note that securely sorting an encrypted array is believed to be harder than secure report; furthermore, in-order insertion to an encrypted array seems to require secure search as a prerequisite.

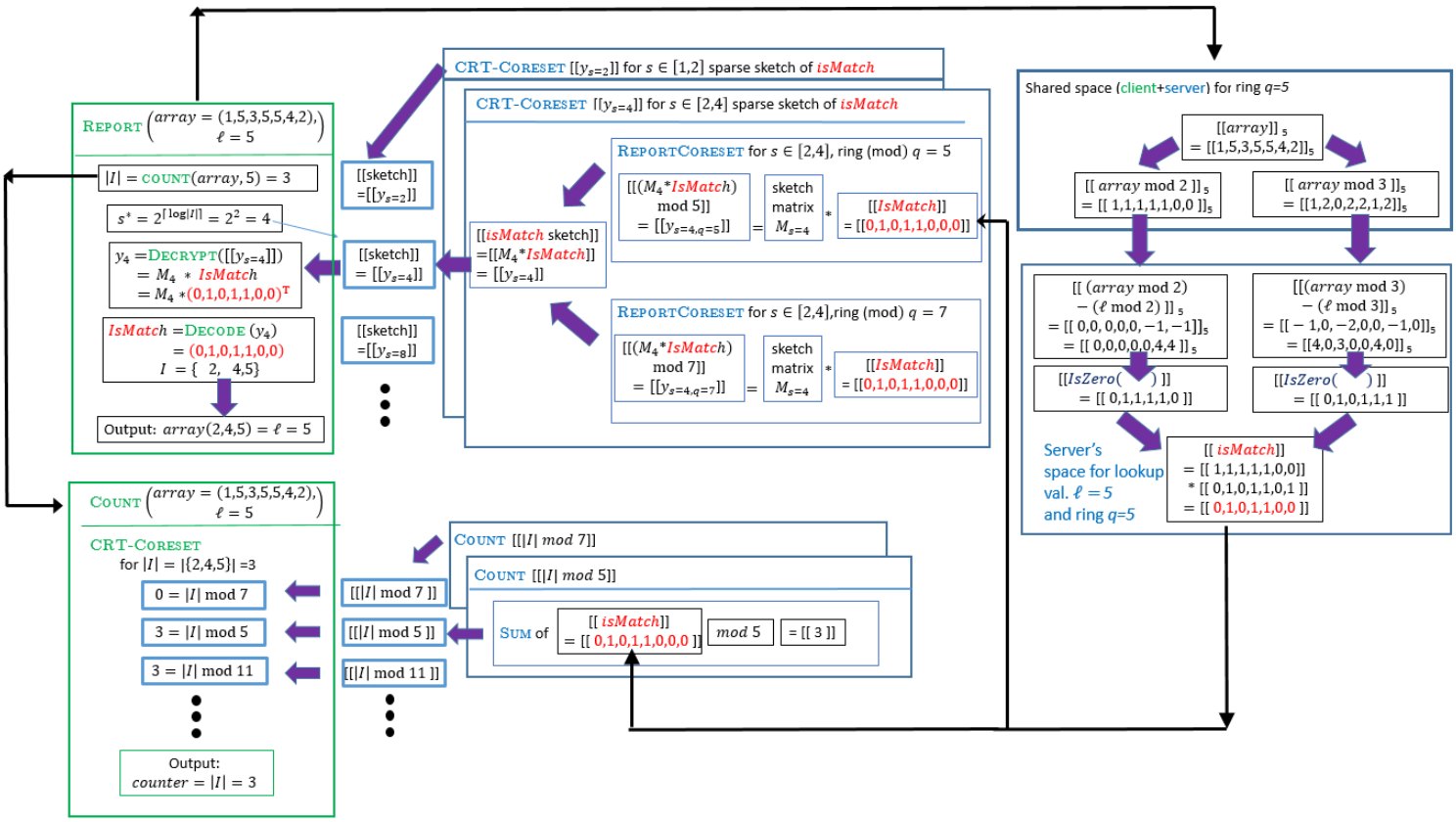

Figure 1: Overview of the suggested system for reporting the indices $I = \{2, 4, 5\}$ of the lookup value $\ell = 5$ in the array $(1, 5, 3, 5, 5, 4, 2)$. Green and blue titles correspond to algorithms that run on the client's and server's side, respectively. The rightmost boxes represent encrypted saved versions of the array. After the client calls REPORT on the right, a version is computed from the shared memory (top right) to each of the few ring values (only $q = 5$ is shown). ISMATCH (bottom right) contains binary results for each ring. The client computes the number $|I|$ of occurrences (bottom left) of ℓ in the array, by applying the CRT-CORESET on the COUNT results that are computed by the server. It then computes the closest power of 2 for $|I|$ (i.e., $s^* = 4$), decrypts the corresponding sketched vector ($y_s = 4$), and ignores the other sketches. The client then compute I efficiently from y_s. Each sketch vector is computed by the server using CRT-Coreset which computes, for every ring's size q, its REPORTCORESET (in the middle) that uses ISMATCH $= array - (\ell, \cdots, \ell)$ for q (bottom right).

client's query, the retrieved data item, and the query and access patterns. The query here is typically the address $i \in [m]$, nevertheless, the above constructions are easily extended for retrieving encrypted keywords provided the keyword is *uniquely identifies* at most a single data item (unique identifier constraint). This is related to PIR-by-keywords [19], but unlike the latter can address both the case of plaintext and encrypted data items, provided uniqueness holds. For settings where the data is not encrypted, indexing can be used to enforce uniqueness (however, with considerable time and memory overhead) [12, 57].

Secure search on encrypted data, eliminating the aforementioned uniqueness constraint, was recently achieved [2]. In this work both data and query are encrypted, and the protocol hides all the following: data items, query, and query and access patterns. Lack of

uniqueness necessitates specifying *which matching item to retrieve*. In [2] the retrieved item is the *first match*; whereas retrieving subsequent items requires further interaction, one-round for each retrieved item.

Other scenarios: relaxed settings. Numerous other related works on secure search are relevant, if relaxing the above settings to allow, for example, $k > 1$ servers, $r > 1$ rounds, communication bandwidth growing with database size m or with complexity of the computed functionality, or information leakage on query and access pattern, etc.

When allowing $k > 2$ non-colluding servers, fast private queries on public data systems were presented in [5, 60].

When allowing $r > 1$ rounds, $\log m$ communication overhead, and client maintaining and updating state – Oblivious RAM (ORAM) [36] enable hiding data access patterns with sub-linear server time (breaking the linear scan barrier of our settings).

When allowing communication bandwidth to grow with the time to compute the search functionality, classical "pre-FHE" secure two-party computation (2PC) techniques can be employed, such as garbled circuits and secret sharing [62, 38].

When allowing information leakage, as in leaking access and query patterns or revealing order, searchable and structured encryption (SE and STE) [58, 13] allow for highly efficient construction, achieving approaching plaintext search time; yet, with questionable security guarantee.

Table 1: Comparison to related work for securely reporting lookup records in a database column as in Definitions 2. We compare only protocols that have efficient communication complexity as defined in Definition 1, so for example, for PIR that means only a constant number of elements can be retrieved.

Algorithm and Papers	Efficient Client	Efficient Server	Multiple Matches	Returns All Matches	Returns Records	Provably Secure	Records Per Min.
Searchable Encryption [6, 58]	✓	✓	✓	✓	✓	×	10^9
PIR [9, 29, 12, 57]	✓	×	×	×	✓	✓	10^6
Secure Pattern Matching [63, 18, 50, 16, 59, 48]	×	✓	✓	×	✓	✓	10^3
PSI [14]	✓	✓	×	×	×	✓	10^6
Secure Search [2]	✓	✓	×	×	✓	✓	10^6
This paper: Direct method	✓	×	✓	✓	✓	✓	10^3
This paper: CHELib	✓	✓	✓	✓	✓	✓	10^9

1.3 Our Contribution

The main goal of this paper is to introduce a novel paradigm, called *coresets for homomorphic encryption* (CHE), that forges a link between data summarization techniques (known as core-sets or sketches) and cryptography. It enabled us to reduce dramatically the theoretical guarantees for both the degree and communication by removing the dependency of the polynomial in the database size. An overview of the specific example application of this paper is shown in Fig. 1.

While it is not clear if FHE versions exist for many classic tasks, such as search and report over an array, our paradigm suggests instead to return a coreset for these problems. Our technique was already used to solve the secure search problems (where the output of the report is too large) in [2], and we expect it to be used for solving many other problems in the future and turns FHE into more practical paradigm.

To this end, we also provide experimental results with corresponding open C++ library code showing that our new algorithms that are based on this paradigm, can run secure

computation on billions of database records compared to hundred-thousands entries via the prior algorithm. Furthermore, as shown by both our analysis and experiments, this size grows near-linearly with the number of machines on the cloud.

The second goal of this paper is to introduce practical yet simple analysis and empirical tools for the applied crypto community and software engineers. To this end, this paper assumes no background in cryptography. Readers that wish to skip the proofs can still learn the pseudo code in this paper or evaluate the companion open code library [53].

More specifically, we suggest the following contributions:

1. New paradigm: coresets for homomorphic encryption (CHE) which proves that techniques that are traditionally used for data summarization, can be used to speed up the running time of secure database queries toward practical homomorphic encryption with both provable and empirical guarantees for their performance. See Section 1.4.

2. Example coresets to demonstrate our paradigm. The first CHE is called a Report Coreset since it is the main tool of our report application below. It combines ideas from Group Testing [31] and Sketch Matrices to allow the secure compression of large sparse vectors (the desired indices) via a low-degree polynomial, such as sketch matrix which are linear operators. See Sections 3 and 4. The second CHE is called CRT-Coreset since its goal is to allow the usage of the Chinese Remainder Theorem on the output.

3. Example application to demonstrate our coresets. We provide the first polynomial-time secure algorithms for answering database report queries of the form "which records contains a specific lookup value".

4. Experimental results. We implemented our algorithms and turned them into a secure database system, with preliminary experimental results. As expected by our analysis, we can now answer report queries on database of few billion entries in few minutes on a single machine, instead of millions taking roughly a day with the prior state-of-the-art. See Section 5.

5. Open Source Library of Coresets for Homomorphic Encryption (CHELIB) based on HElib [41] is provided for the community, to reproduce our experiments, to extend our results for real-world applications, and for practitioners at industry or academy that wish to use these results for their future papers or products. See [3].

We hope that our algorithms and software library will not only be extended to a general secure SQL open library in the near future by future papers, but also forge a link between the cryptography and data summarization experts.

1.4 New Paradigm: Coresets for Homomorphic Encryption (CHE)

Coreset is a data summarization C of a set P of items (e.g. points, vectors or database records) with respect to a set Q of queries (e.g. models, shapes, classifiers, points, lines) and a loss function f, such that $f(P,q)$ is approximately the same as $f(C,q)$ for every query $q \in Q$. The goal is to have provable bounds for (i) the size of C (say, $1/\varepsilon$), (ii) the approximation error (say, $\varepsilon \in (0,1)$) and (iii) construction time of C given (P,Q,f). We can then run (possibly inefficient) existing algorithms and heuristics on the small coreset C, to obtain provably approximated solution for the optimal query (with respect to f) of the original data.

Sketches are a special type of coresets, where given an $m \times d$ matrix P and a "fat", $s \times m$ matrix $A_{s,m}$, with $s \ll m$, the result $C = A_{s,m}P$ is a $s \times d$ ("sketch") vector. Many

problems can be solved on the sketch C instead of the long vector P, by designing a corresponding (sketch) matrix S. For example, if the entries of S are random standard Gaussian variables, then we can approximate the k-means problem on C by the Johnson-Lindestrauss Lemma; see [7]. In this paper we use sketches in the context of compressed sensing, or more precisely, Group Testing [31].

The coreset is a paradigm in the sense that its exact definition, structure and properties change from paper to paper. The term "coreset" was coined by Agarwal, Har-Peled, and Varadarajan [1] and originally used for optimization in computational geometry (e.g. [54, 21, 11]). Since then coresets were used in fields such as machine learning (e.g.[44, 10]), numerical algebra [61], graph theory (e.g. [27]), time series [45, 55, 35] streaming [24, 32, 22, 25] and pattern recognition [15, 23, 40, 26, 43].

Coresets for Homomorphic Encryption (CHE). In this paper we suggest a new paradigm that is inspired by the coreset paradigm, but for a very different application and context. Instead of running the complete secure algorithm on the server side and encrypted data, the server computes only (encrypted) coresets for the problem at hand and sends them back to the client. Since the coresets are small, communicating them to the client, decrypting them, and decoding the desired result from the coreset is relatively fast and require only little additional time on the client side.

In this paper we introduce the first two CHEs: one for reporting non-zero values in an array, and one for reducing the ring size from r to $O(\log r)$; see Sections 3 and 4 respectively. These two coresets are then combined to give our secure solution to the report problem; see Theorem 1. The report coreset uses a sketch matrix to communicate the sparse binary vector of the reported indices to the client.

The key property that we show in this paper is that computing coresets on the server side securely (i.e., via polynomials) dramatically reduces its computation by reducing the degree of the polynomial from at least linear to logarithmic on the input size (CRT coreset) or even constant (report coreset).

Informally, in this paper a function S is an efficient coreset construction scheme for a problem if it has the following pair of properties.

(i) $S(P) \in \mathbb{R}^k$ is a short vector that can be computed efficiently. Usually $k = (\log m)^{O(1)}$ for an input vector P of length m.

(ii) $S(P)$ is a core-set or a "sketch" vector in the sense that the desired information, in a problem dependent sense, can be extracted efficiently and exactly from $S(P)$, without having access to (the large) original vector.

Another significant difference from traditional coreset and sketches papers is that the coresets in this paper are exact in the sense that the data reduction does not introduce any additional error ε.

Group Testing meets FHE for report coreset. Our report coreset in Protocol 3 uses a main modern result from the field of Group Testing [31]. To our knowledge, this is the first application of Group Testing to FHE. The motivation is to return to the client the very large (m bits) but sparse binary (indicator) vector χ of the desired s indices that were computed in Line 5 of our report coreset (Protocol 3) by communicating number of bits that is only poly-logarithmic in m.

Since the vector is encrypted, an efficient algorithm must be realized by a low-degree polynomial, so simple comparison of each entry to 1 is impossible by such a polynomial. Instead, Group Testing suggests to extract the set of 1's entries in χ from a small number of predefined sums on subsets entries in χ.

In the context of FHE, our Report Coreset multiplies the input vector by a (sketch) matrix $A_{s,m}$ of size $k \times m$ where k is polynomial in s and $\log m$, i.e., we compute few *linear* combinations $A_{s,m} \cdot \chi$ of the entries. Indyk, Ngo and Rudra relatively recently [46]

suggested such a binary matrix $A_{s,m}$, so that our χ vector can be decoded from $A_{s,m} \cdot \chi$ in time that is also polynomial in s and $\log m$ on the client side. Unfortunately, unlike χ, the vector $A_{s,m} \cdot \chi$ is not a binary vector and may be more efficient to compute over ring with larger size $p > 2$. Looking ahead, if isMatch() uses Fermat's little theorem for making the equality test, this results in a high-degree polynomial for computing χ in Line 5 of Protocol 3. In addition, the value s, the sparsity of χ is required to compute $A_{s,n}$ but unknown to the server. These and other challenges are handled by the CRT-coreset, and the other algorithms of our system.

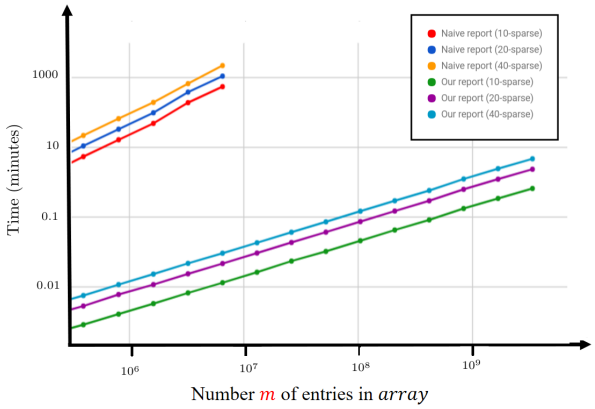

Figure 2: Server's running time (y-axis) on a single machine on Amazon's cloud, for different vector size (x-axis) of Secure Report (Protocol 3) over encrypted vector. Unlike the direct method, the running time of the more involved method reduced linearly with number of machines. Each colored curve represents a test with a vector of different sparsity s. The red dots represent actual experiments.

Figure 3: Running time (y-axis) of our system using multiple machines (in color) on the cloud for different number of database records (x-axis). As implied by the analysis, the running time reduces linearly with number of machines, unlike existing secure implementations or the direct method that do not use coresets.

2 Problem Statement and Main Theorem

2.1 The Secure Report Problem

Definition 2 (The Report Problem)**.** Let $array = (x_1, \ldots, x_m)$ be a vector of m (not necessarily sorted) integers between 0 and $r-1$, isMatch() a predicate and ℓ a full specification of the predicate defining a match. An algorithm *solves the Report problem* if it gets a pair $(array, \ell)$ as input, and returns the set I of all occurrences in $array$ matching ℓ, i.e.,

$$I = \{\, (i, x_i) \mid i \in \{1, \cdots, m\}, \textsc{isMatch}(x_i, \ell) = 1 \}.$$

The Secure-Report problem is the Report problem where the input data as well as the output are replaced by their encryption versions. That is, the input are ciphertexts $[\![array]\!]$ and $[\![\ell]\!]$ which are the encryption of $array$ and ℓ respectively. The output is a ciphertext $[\![I]\!]$ which is the encryption of the desired outcome I.

In the context of our coreset paradigm, the output is a short sketch, named, *Report Coreset*, on which the client applies an efficient decoding algorithm to obtain all the occurrences I where $\textsc{isMatch}(array(i), \ell) = 1$.

Definition 3 (Report Coreset)**.** Let k, m, $array$, ℓ and I be as in Definition 2. A vector $y \in \mathbb{Z}^k$ is a *k-report coreset* for $(array, \ell)$ if, given only y, we can decode (compute) the set I of all occurrences of the lookup value ℓ in $array$.

The usage scenario is that the server computes $[\![y]\!] = [\![f(array, \ell)]\!]$ while seeing encrypted values only, whereas the client decrypts and decodes to obtain the desired outcome I as in the next formal definition.

2.2 Main Theoretical Result

Based on Definitions 2 we can now define the following problem statement:
Can we securely solve the Report Problem with a server realizing a polynomial with a degree sub-linear in the size m of the input $array$? Can we do this via a sub-linear communication?

In this paper we answer both of these questions affirmably.

In contrast, the existing solutions to the Report Problem as defined in this paper have a server realizing a polynomial whose degree is $\Omega(m)$ or having communication complexity $\Omega(m)$.

Specifically, the following theorem is an immediate corollary of our solution to the Report Problem; see Section 3:

Theorem 1 (Secure Report)**.** *Protocol 3 provides a one round protocol that securely solves the report problem on $array \in \{0, \ldots, r-1\}^m$ with a predicate* isMatch() *which is a polynomial of degree* $\deg(\textsc{isMatch})$*, and a predicate specification ℓ, with a server evaluating a polynomial of degree* $\deg(\textsc{isMatch})$*, client with running time $(|I| \cdot \log m)^{O(1)}$ and communication complexity $O(|I| \cdot \log m + |\ell|)$, for $I = \{\, (i, x_i) \mid i \in \{1, \ldots, r\}, \textsc{isMatch}(x_i, \ell) = 1 \}$.*

More applications. While our main motivation and application is secure report, our main technical result is of independent interest: how to search an array via a polynomial of low-degree that can be computed in parallel and using only simple operations. We hope that this result will be used for many other applications in the future such as computations on GPUs which usually support limited set of operations that includes multiplications and additions, or low energy IoT ("Internet of Things") devices with limited computation power and memory (e.g., cannot run software libraries such as Matlab) that may still be able to compute such polynomials efficiently on board, or alternatively, send the request to the cloud via their internet connection.

3 The Simple Protocol

In this section we describe a protocol for the report problem. We first start with a simple version that assumes *cnt* (the output size) is known, and uses a large enough ring size $p > cnt$. In Section 3.1.2 we discuss how to remove the assumption that *cnt* is known, and in Section 4 we discuss how the ring size can be reduced.

Reduction to Binary Vector The output of the protocols we describe below is a binary vector $\chi = \{0,1\}^m$, such that $\chi_i = \text{isMatch}(x_i, \ell)$ (i.e. $\chi_i = 1$ if x_i matches the predicate and 0 otherwise). By the properties of the sketch matrix $A_{cnt,m}$, the set $I' = \{\, i \mid i \in \{1, \cdots, m\}, \text{isMatch}(x_i, \ell) = 1\}$. can be reconstructed from $A_{cnt,m}\chi$.

Similarly, we consider the bit representation $x_i(1), \ldots, x_i(\lceil \log_2 r \rceil)$ of x_i (we recall that $x_i \in \{0, \ldots, r-1\}$) and compute $\lceil \log_2 r \rceil$ sketches $A_{cnt,m}\chi(j)$, where $\chi_i(j) = \chi_i x_i(j)$, from which $I'' = \{\, x_i \mid i \in \{1, \cdots, m\}, \text{isMatch}(x_i, \ell) = 1\}$ can be computed.

Putting it all together, we can construct a protocol whose output is

$$I = \{\, (i, x_i) \mid i \in \{1, \cdots, m\}, \text{isMatch}(x_i, \ell) = 1\}.$$

Protocol Overview.

For robustness, we consider three parties: the Cloud who offers storage and computation services, the Data Source(s) that upload data to the Cloud and the User who performs queries on the data stored in the Cloud. We provide secure protocols for the Data Source to upload data to the Cloud and for the User to query it where the cloud is unable to read the data. We describe: (i) an initialization protocol, (ii) an upload protocol and (iii) the report protocol where the User submits a filter to the Cloud who then reports efficiently all data records matching the filter.

We now describe the three protocols in more details.

Initialization The initialization protocol is described in Protocol 1. The User chooses a ring size $p > cnt$ that ensures the correctness of the output (specifically, Protocol 3, Line 6). In the simple protocol we assume *cnt* is known. After setting p appropriately the User initializes the keys in line 3 and sends the public keys to the Cloud and to the Data Source (Line 4).

Upload The upload protocol is described in Protocol 2. The Data Source encrypts the data (Line 2) with the public key and sends it to the Cloud. The encryption can be done in parallel and may be distributed among many data sources. Notice that the Data Source does not have the secret key, so even if one Data Source is compromised, the privacy of data uploaded from other Data Sources or historical data uploaded from the compromised source is not compromised.

Report In the report protocol (Protocol 3), the User sends a filter given by a predicate isMatch() and ℓ to the Cloud and receives a sketch for an indicator vector $(\text{isMatch}(x_1, \ell), \ldots, \text{isMatch}(x_m, \ell))$. In the simple case, isMatch() is agreed upon in advance, in which case the User sends only ℓ to the Cloud. For example, $\text{isMatch}(a, \ell)$ can be the equality, i.e. $\text{isMatch}(a, \ell) = 1$ iff $a = \ell$. In the general case, isMatch() can be determined ad-hoc and be sent by the User to the Cloud together with the parameter ℓ. The parameter ℓ is encrypted (Protocol 3, Line 2) before being sent to the Cloud. The Cloud then applies isMatch on all data records $x_1, \ldots, x_m$ independently in parallel, the output of which is kept in an indicator vector $\chi \in \{0,1\}^m$ (Line 5). In this first implementation of the protocol we assume *cnt* (sparsity of χ) is known. Knowing *cnt* we construct (in offline) a binary (cnt, m) sketch matrix, $A_{cnt,m}$. The Cloud computes $A_{cnt,m}\chi$, which compactifies the m-dimensional vector χ into a $O(cnt \log m)$-dimensional vector χ' (Line 6), which is then sent to the User (Line 7). By the properties of $A_{cnt,m}$ (see Section 1.4) the User can reconstruct χ from χ' since χ is *cnt*-sparse (Line 10).

Protocol 1: Initialization

Inputs:
User: has a security parameter λ and integers $cnt < m$,
Outputs:
User: gets a key (Pk, Sk, Ek).
Cloud: gets (Pk, Ek).
Data Source: gets (Pk, Ek).
Shared Parameters:
An FHE scheme $E = (Gen, Enc, Dec, Eval)$.

User performs:
choose a prime $p > cnt$.
generate keys $(sk, pk, ek) := Gen(1^\lambda, p)$.
send p, cnt, m, pk, ek to *Cloud* and to *Data Source*.

Protocol 2: Upload

Inputs:
Data Source: a vector $array = (x_1, \ldots, x_m)$.
Outputs:
Cloud: gets m and the encrypted vector
$[\![array]\!] = ([\![x_1]\!], \ldots, [\![x_m]\!])$.

Data Source performs:
compute $[\![x_i]\!] := Enc(x_i, pk)$, for $i = 1, \ldots, m$
send $array$ to *Cloud*.

3.1 Implementation Details

3.1.1 Implementations of isMatch

Implementing isMatch as equality.

As a first naive implementation of $\textsc{isMatch}(a, b)$ we consider the equality test where $a, b \leq r$ are given as bits, i.e. $(a_1, a_2, \ldots)$ are the bits of a, where each $a_i \in \{0, 1\}$, the i-th bit of a, is encrypted in a different ciphertext and similarly $(b_1, b_2, \ldots)$ are the bits of b. In that case, it can be given as a simple polynomial

$$\textsc{isMatch}(a, b) = \textsc{IsEqual}(a, b) = \prod(1 - (a_i - b_i)^2),$$

The degree of this polynomial is $2\log_2 r$ and it can be realized with $2\log_2 r - 1$ MULT operations. We note that a simpler polynomial exists for equality testing over bits, $\textsc{IsEqual2}(a, b) = \prod(a_i + b_i + 1) \mod 2$. Implementing this polynomial in an arithmetic circuit is efficient only when the circuit is over a ring of size $p = 2$ and the modulo operation is inherent from the ring size. However, in many implementations the circuit is designed over a larger ring size $p > 2$. Specifically, computing $[\![\chi']\!] = A_{cnt,m}[\![\chi]\!]$ is significantly more efficient when $p > 2$. In that case, it makes sense to keep a (similarly b) as a base p number given by its digits. $a = (a_1, a_2, \ldots)$, where $a_i \in \{0, \ldots, p-1\}$ is the i-th digit in base p. In that case, assuming p is prime and using Fermat's little theorem we can set

$$\textsc{isMatch}(a, b) = \prod(1 - (a_i - b_i)^{p-1}) \mod p,$$

which has a slightly better degree $\frac{\log r}{\log p}(p-1)$ and can be realized with $O(\log r)$ MULT operations.

Protocol 3: Secure Report

Inputs:
User has the key (sk, pk), a value ℓ and a predicate isMatch.
Cloud has the public and evaluation keys pk, ek, and an encrypted array $[\![array]\!]$.
Shared Parameters:
An FHE scheme $E = (Gen, Enc, Dec, Eval)$,
Integers $m, p > cnt$.
Output:
The *User* output is an indicator vector χ of indices in $array$ matching ℓ.

1 **The User performs:**
2 compute $[\![\ell]\!] := Enc(\ell, pk)$.
3 send $[\![\ell]\!]$ to the *Cloud*.
4 **The Cloud performs:**
5 compute $[\![\chi(i)]\!] := \text{isMatch}([\![x_i]\!], [\![\ell]\!])$, in parallel.
/* A_{cnt} `is binary` (cnt, m)`-sketch matrix.` */
6 compute $[\![\chi']\!] := A_{cnt,m}[\![\chi]\!]$.
7 send $[\![\chi']\!]$ to *User*.
8 **The User performs:**
9 decrypt $\chi' := Dec([\![\chi']\!], sk)$.
10 decode $\chi := Decode(\chi')$.

Implementing non-trivial isMatch filters. The $\text{isMatch}(a, b)$ can have non-trivial implementations, for example, a can be a vector, b a range and $\text{isMatch}(a, b) = 1$ if $a \in b$. Thus Protocol 3 can report elements matching any filter. The depth of the circuit realized by the Cloud in Protocol 3 is asymptotically equal to the depth of the circuit realizing isMatch, since all instances of isMatch are realized in parallel.

3.1.2 Computing *cnt*

In Protocol 3 we assumed cnt, the number of items in $array$ matching ℓ is known. We now show how to remove this assumption. We first assume that cnt can be shared with the cloud. In that case, Protocol 4 can be run as a preliminary step, which computes cnt. The value cnt can then be shared with the Cloud and Protocol 3 then continues as before, increasing the number of rounds in the solution from 1 to 2.

Protocol 5 deals with the case where cnt cannot be shared with the Cloud. In this case, the Cloud sets $s = 2^0, 2^1, \ldots, m$, computes $\log_2 m$ sketches and sends them to the client. Although the Cloud has communication complexity of $O(m)$, the user can abort the protocol after receiving only $O(cnt)$ bits.

4 Protocol with Reduced Ring Size

The ring size required by Protocol 3 and Protocol 4 as described in Section 3 is $p > s$ to allow a correct computation of $A_s \cdot \chi$ (Line 6 in Protocol 3 and Line 6 in Protocol 4).

In many cases, the Chinese Remainder Theorem (CRT) can be used to replace a large ring size with many smaller ones. Unfortunately, CRT cannot always be used. Specifically, it cannot be used if the isMatch circuit implements an equality test based on Fermat's

Protocol 4: Secure Count

Inputs:
User has the key (sk, pk), a predicate isMatch() and its specification ℓ.
Cloud has the public and evaluation keys pk, ek, and an encrypted array $[\![array]\!]$.
Shared Parameters:
An FHE scheme $E = (Gen, Enc, Dec, Eval)$, the number m, and a prime $p > m$.
Output:
The output of *User* is a number $cnt = |\{i \mid \text{isMatch}(array(i), \ell) = 1\}|$.

1 **The User performs:**
2 compute $[\![\ell]\!] := Enc(\ell, pk)$.
3 send $[\![\ell]\!]$ to the *Cloud*.
4 **The Cloud performs:**
5 compute $[\![\chi(i)]\!] := \text{isMatch}([\![array(i)]\!], [\![\ell]\!])$, in parallel.
6 compute $[\![cnt]\!] := \sum [\![\chi(i)]\!]$.
7 send $[\![cnt]\!]$ to *User*.
8 **The User performs:**
9 decrypts $cnt := Dec([\![cnt]\!], sk)$.

Little Theorem (FLT) (which is a useful and popular [2] primitive). In a nut shell, a FLT-based-equality is realized by $\text{IsEqual}(z) = z^{p-1} \mod p$ which is a *different polynomial* for each moduli p, whereas CRT works when computing the *same polynomial for all moduli*.

In this section we consider modifications of Protocol 3 and Protocol 4 that use many small ring sizes with conjunction of a FLT-based isMatch. For simplicity we consider the case where $\text{isMatch}(a, b)$ is a simple equality test $a = b$.

Using the Chinese Remainder Theorem (CRT) with an equality test that does not involve FLT we run Protocol 4 with $\kappa = \frac{\log m}{\log \log m}$ different ring sizes $\log m < p_1 < \ldots < p_\kappa$. The outputs of the κ copies of Protocol 4 are $(cnt \mod p_1), \ldots, (cnt \mod p_\kappa)$, from which cnt can be reconstructed. Accordingly, Protocol 1 should be changed to initialize κ keys, and Protocol 2 should encode $array$ with κ keys. In addition Protocol 3 should be run κ times, each having an appropriate version of the isMatch oracle.

An equality test that does not involve FLT, can be for example bitwise comparison, $\text{isMatch}(a, b) = 1$ iff $a = b$, where $a, b < M$ and a_i (resp. b_i) is the i-th bit in the binary representation of a (resp. b). In that case, we can take $\text{isMatch}(a, b) = \text{IsEqual}(a, b) = \prod 1 - (a_i - b_i)^2$ which is a $2 \log M$ degree polynomial and is evaluated correctly over all moduli p_i. The communication complexity of the upload protocol, then becomes $O(m \frac{\log m \log M}{\log \log m})$ since it needs to encrypt the m values of $array$ as $\log M$ bits with $\frac{\log m}{\log \log m}$ keys.

Motivated to reduced the communication complexity we use a slightly different representation for $array$.

Definition 4 (CRT Representation)**.** Let $a < M$ be an integer, $\mu = \frac{\log M}{\log \log M}$ and $\log M < q_1 < \ldots < q_\mu$ be primes. The *CRT representation* of a is $(\overline{a})_{q_1 \ldots q_\mu} = (a \mod q_1, \ldots, a \mod q_\mu)$. When $q_1, \ldots, q_\mu$ are obvious from the context we simply write $\overline{a}$.

Lemma 1. *Given two integers* $0 \leq a, b \leq M$ *and primes* $\log M < q_1 < \ldots < q_\mu$*, where* $\mu = \frac{\log M}{\log \log M}$*, then* $a = b$ *iff* $(a = b) \mod q_j$ *for all* j*.*

Protocol 5: Secure Report where cnt is secret

Inputs:
User has the key (sk, pk), a value ℓ, a bivariate function ISMATCH,
s.t. ISMATCH$(x, y) = 1$ if $x = y$ and 0 otherwise
Cloud has the public and evaluation keys pk, ek,
and an encrypted array $[\![array]\!]$.
Parameters, Outputs: Same as Protocol 5

The User performs:
  compute $[\![\ell]\!] := Enc(\ell, pk)$.
  send $[\![\ell]\!]$ to the *Cloud*.
The Cloud performs:
  compute $[\![\chi(i)]\!] :=$ ISMATCH$([\![array(i)]\!], [\![\ell]\!])$, in parallel.
  for each $s := 2^1, 2^2, \ldots, m$ **do**
    `/* A_s is binary (s,m)-sketch matrix */`
    compute $[\![\chi'_s]\!] := A_s[\![\chi]\!]$. send $[\![\chi'_s]\!]$ to *User*.
The User performs:
  ignore A_s where $s < cnt$ or $s > 2cnt$
  decrypt $\chi'_s := Dec([\![s]\!], sk)$, where $cnt < s < 2cnt$

The proof follows immediately from the Chinese Remainder Theorem since $\prod q_j > M$.

For simplicity, assume first that $q_j < p_i$ for any i, j, this can always be made true since we have only a lower bound in choosing q_j and p_i. The ISMATCH oracle, can be implemented using Fermat's Little Theorem: $f_{p_i}(\overline{a}, \overline{b}) = \prod(1 - (a_j - b_j)^{p_i - 1})$, where $\overline{a}$ and $\overline{b}$ are the CRT-representation of a and b respectively. Then f_{p_i} is a polynomial of degree $O(\log N \log M)$.

To address cases where $q_j > p_i$ we represent values of $array \mod q_j$ as multi-digits numbers in base p_i, each number having at most $O(\frac{\log q_j}{\log p_i}) = O(\frac{\log \log M}{\log \log N})$ digits.

We therefore describe in full a more economic (in communication complexity) report protocol for the case where ISMATCH is the equality implemented using FLT.

Protocol 6: Initialization

Inputs:
User: has a security parameter λ and an integer m,
where we assume w.l.o.g. m is a power of two
Outputs:
User: gets $\kappa = \frac{\log m}{\log \log m}$ keys
$(Pk_1, Sk_1, Ek_1), \ldots, (Pk_\kappa, Sk_\kappa, Ek_\kappa)$.
Cloud: gets $(Pk_1, Ek_1), \ldots, (Pk_\kappa, Ek_\kappa)$.
Data Source: gets $(Pk_1, Ek_1), \ldots, (Pk_\kappa, Ek_\kappa)$.
Shared Parameters:
An FHE scheme $E = (Gen, Enc, Dec, Eval)$.
User performs:
  choose μ primes $\log m < q_1 < \ldots < q_\mu$.
  choose κ primes $q_\mu < p_1 < \ldots < p_\kappa$.
  generate κ keys $(sk_1, pk_1, ek_1) := Gen(1^\lambda, p_1) \ldots (sk_\kappa, pk_\kappa, ek_\kappa) := Gen(1^\lambda, p_\kappa)$.
  send m, $q_1, \ldots, q_\mu$, $p_1, \ldots, p_\kappa$, $(pk_1, ek_1), \ldots, (pk_\kappa, ek_\kappa)$ to *Cloud* and to *Data Source*.

Protocol 7: Secure Upload

Inputs:

Data Source: a vector $array = (array(1), \ldots, array(m))$, where we assume w.l.o.g. m is a power of two

Outputs:

Cloud: gets m and $\kappa \times \mu$ encrypted vectors $[\![array_j]\!]_i = ([\![array(1) \mod q_j]\!]_i, \ldots, [\![array(m) \mod q_j]\!]_i)$, for $1 \leq i \leq \kappa$ and $1 \leq j\mu$.

1 **Data Source performs:**
2 **for** *every* $1 \leq i \leq \kappa$ *and every* $1 \leq j \leq \mu$ **do**
3 compute $[\![array(k)_j]\!]_i := Enc(array(k) \mod q_j, pk_i)$, for $k = 1, \ldots, m$
4 send $[\![array_j]\!]_i$ to *Cloud*.

4.1 Protocol Overview

Protocol 8 is based on Protocol 3. Unlike the latter, this protocol supports only filter oracles isMatch that implements equality tests using FLT. The protocol starts by encrypting $array(i) \mod q_i$, for different primes q_i and using different keys with different ring sizes (Line 3). The multiplicity of moduli and ring sizes, lets the protocol determine whether $array(i) = \ell$ by comparing many (significantly smaller) moduli (by using f). This comparison is done for all keys, in their respective ring sizes. Then the protocol computes the report sketch using different keys with different ring sizes (Line 7).

The sketch $[\![\chi']\!]_i$ is computed with respect to the i-th key and modulo its ring size p_i. By the CRT, given $\chi \mod p_i$, for $i = 1, \ldots, \kappa$, we can reconstruct χ' (as done in Line 11). The indicator vector χ can be reconstructed from χ' as in Protocol 3.

Note, that the CRT-coreset described here is a general framework to compute sketches with small ring sizes while still using FLT. The two sketches we use here: counting and reporting are obtained by replacing Line 7 with the appropriate computation.

5 System & Experimental Results

In this section we describe experiments on the secure report that we implemented based on our algorithms. To our knowledge, this is the first implementation of such a secure report system. For example, it can report the locations of 1's in a 10-sparse vector of size $3 \cdot 10^9 = 3,000,000,000$ entries in less than one minute by using a single machine on Amazon EC2 cloud. The system is fully open sourced, and all our experiments are reproducible. We hope to extend and improve the system in future papers together with both the theoretical and practical community.

5.1 The System

System Overview. The system maintains an encrypted database that is stored on Amazon's AWS cloud. The system gets from the client an encrypted lookup value ℓ to search for, and a column name $array$ in a database table of length m. The encryption is computed on the client's side using a secret key that is unknown to the server. The client can send the request through a web-browser, that can be run e.g. from a smart-phone or a laptop. The system then runs our secure report coreset algorithm on the cloud, and returns a report coreset for $(array, \ell)$. The web browser then decrypts this coreset on the client's machine and uses it to compute the solution to the report query, which is the

Protocol 8: Secure Report

Inputs:
User has κ keys $(sk_1, pk_1) \dots (sk_\kappa, pk_\kappa)$, a value ℓ
and a function isMatch as in Protocol 3
Cloud has the public and evaluation keys
$(pk_1, ek_1), \dots (pk_\kappa, ek_\kappa)$,
and $\kappa \times \mu$ copies of array
$[\![array \mod q_j]\!]_{p_i}$, for $1 \le i \le \kappa$ and $1 \le j \le \mu$.
Shared Parameters:
An FHE scheme $E = (Gen, Enc, Dec, Eval)$,
the number m, primes $\log m < p_1 < \ldots < p_\kappa$ and
$\log m < q_1 < \ldots < q_\mu$.
Output:
The *User* output is an indicator vector χ
of indices in *array* matching ℓ.
Assumption:
A bound $|\chi| < s$ is known to all parties.

The User performs:
compute $[\![\ell \mod q_j]\!]_i := Enc(\ell, pk_i)$, for all $1 \le i \le \kappa$ and $1 \le j \le \mu$.
send $[\![\ell \mod q_j]\!]_i$ to the *Cloud*.
The Cloud performs:
compute $[\![\chi(i)]\!]_i := f(\{[\![array(k \mod q_j)]\!]_{p_i}\}_{j=1}^{\mu}, \{[\![\ell \mod q_j]\!]_{p_i}\}_{j=1}^{\mu})$, in parallel.
compute $[\![\chi']\!]_i := A_s[\![\chi]\!]$. `//` A_s `is binary` (s, m)`-sketch matrix`
.
send $[\![\chi']\!]_i$ to *User*.
The User performs:
decrypt $\chi'_i := Dec([\![\chi']\!], sk_j)$.
CRT decode χ' from $\chi'_1, \dots, \chi'_\mu$
decode $\chi := Decode(\chi')$.

indices $i_1, \dots, i_s$ in *array* that contains ℓ. Database updates can be maintained between search calls, and support multiple users that share the same security key.

Hardware. Our system is generic but in this section we evaluate it on Amazon's AWS cloud. We use one of the standard suggested grids of EC2 x1.32xlarge servers, each with 64 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) cores and 1,952 GigaByte of RAM.

Open Software and Security. The algorithms were implemented in $C++$. HELib library [41] was used for the FHE commands. The source of our system is open under the GNU v3 license and can be found in [3]. For our experiments below we use a security key of 80 bits of security.

5.2 Experimental Results

In this sub-section we describe our preliminary experiments with our system and explain the results. Due to lack of space we omit more results and description that can be found in the fuller version [4].

Data. We ran the system on a lookup value $\ell = 1$ in a *array* of m integers of range $r = m$. The vector was all zeroes except for s random indices, and different values for m and s were used. As expected, the actual values behind the encrypted records had no effect on the running times.

Protocol 9: Simple Secure Count

Inputs: same as in Protocol 8

Output:

The output of *User* is a number $cnt = |\{i \mid \text{ISMATCH}(array(i), \ell) = 1\}|$.

The User performs:

compute $[\![\ell \mod q_j]\!]_i := Enc(\ell, pk_i)$, for all $1 \le i \le \kappa$ and $1 \le j \le \mu$.

send $[\![\ell \mod q_j]\!]_i$ to the *Cloud*.

The Cloud performs:

compute $[\![\chi(i)]\!]_i := f(\{[\![array(k \mod q_j)]\!]_{p_i}\}_{j=1}^{\mu}, \{[\![\ell \mod q_j]\!]_{p_i}\}_{j=1}^{\mu})$, in parallel.

compute $[\![s]\!]_i := \sum_k [\![\chi(k)]\!]_i$.

send $[\![s]\!]_i$ to *User*.

The User performs:

decrypts $s_i := Dec([\![s]\!]_i, sk_i)$.

decode $s := CrtDecode(s_1, \ldots, s_\kappa)$.

The Experiment. We ran Algorithm 3 for database table columns ranging from $m = 10$ to
$m = 3,000,000,000 = 3 \cdot 10^9$ records, and $s \in \{10, 20, 40\}$.

Results. Our experimental results for a single machine on the cloud are shown as the circle points in Fig 2. The table of exact values appears in the full version [4].

The client's decoding time was negligible in all the experiments, so the server's time equals to the overall running time. For example, the graph shows that a single machine can report in about 2 minutes all 1's in a 20-sparse column of 3,000,000,000 binary entries.

Comparison to naive approach. Our theoretical results proves that the running time of our new algorithm is only poly-logarithmic in the number m of entries compared to the naive approach which is polynomial in m. However, it assumes that both our and the naive algorithm may use m machines in parallel, i.e., a machine for each record. The goal of our experiment was to show a significant time reduction even using a single machine on the cloud where a running time that is linear in m is expected.

The graph in Fig. 2 is log scaled so a linear curve shows a polynomial relation, and its slope is the degree of the polynomial. Our experiments indeed show that our algorithm is linear in m as expected. The naive algorithm perform worse, both on small number of entries (the ratio is approximately 1000), but also asymptotically: The slope of the naive approach is about 1.5, indicating a running time of $O(m^{1.5})$ (as oppose to our $O(m)$). The main reason we see from the profiling is that, unlike the naive algorithm, our algorithm does not use multiplications beyond the reduction to binary indicator vector, since we implemented the binary sketch matrix using sum of subsets of entries.

A Fundamental Definitions

Definition 5 (Universal constant)**.** A function, data structure or a parameter in an algorithm is a universal (global) constant if it is independent of the content of the corresponding input data. It may still depend on the size of the data.

For example, a universal constant may depend on the input size such as r and m in the Report Problem, but not on the dynamic input values such as input *array* or lookup value ℓ. We do not need to compute a universal constant as part of our main algorithm. Instead, we can compute it only once and, e.g., upload it to a public web-page. We can thus ignore

Table 2: Server's running time of Report Coreset (Algorithm 3) as measured on a single machine on Amazon's cloud for different database *array* size (left column) and different sparsity over encrypted database. The $2^{nd}, 3^{rd}$ and 4^{th} columns show the time in minute to compute report with the naive algorithm with $s = 10, 20$ and 40. The $5^{th}, 6^{th}$ and 7^{th} columns show the time in minute to compute report with our algorithm with $s = 10, 20$ and 40.

m (vector size)	Naive (min)			Our Alg. (min)		
	$s = 10$	$s = 20$	$s = 40$	$s = 10$	$s = 20$	$s = 40$
89,600	1.00	1.99	3.95	0.00	0.00	0.00
192,000	1.68	3.40	6.76	0.00	0.00	0.00
396,800	5.45	11.07	22.17	0.00	0.00	0.00
806,400	16.68	33.63	67.59	0.00	0.00	0.00
1,625,600	49.41	99.81	198.64	0.00	0.00	0.00
3,264,000	195.22	392.71	689.37	0.00	0.00	0.00
6,540,800	564.37	1,133.61	2,281.18	0.00	0.00	0.01
13,094,400				0.00	0.01	0.02
26,201,600				0.01	0.02	0.04
52,416,000				0.01	0.04	0.07
104,844,800				0.02	0.07	0.15
209,702,400				0.04	0.15	0.29
419,417,600				0.08	0.29	0.58
838,848,000				0.17	0.62	1.24
1,677,708,800				0.34	1.22	2.45
3,355,430,400				0.66	2.36	4.72

its construction time in our main algorithm, by passing the constant or a pointer to this constant as an additional input.

Definition 6 (Ring $\mathbb{Z}_p$)**.** The *ring* $\mathbb{Z}_p$ is the set $\{0, \cdots, p-1\}$ equipped with multiplication $(\cdot)$ and addition $(+)$ operations modulo p, i.e., $a \cdot b = \big((a \cdot b) \mod p\big)$ and $a + b = \big((a + b) \mod p\big)$ for every $a, b \in \mathbb{Z}_p$.

Theorem 2 (Follows from CRT [28])**.** *Let $m, k \geq 1$ be integers such that $k \geq \log_2 m / \log_2 \log_2 m$. Let $q = (q(1), \cdots, q(k))$ be a vector of k distinct primes larger than $\log_2 m$. Then there is a unique integer $Alg(m)$ such that $Alg(j) = Alg(m) \mod q(j)$ for every $j \in [k]$. Moreover, $Alg(m)$ can be decoded (computed) from $y = (Alg(q(1)), \cdots, Alg(q(k)))$ in $O(k)$ time, after pre-processing of $k^{O(1)}$ time for computing universal constants.*

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

[2] A. Akavia, D. Feldman, and H. Shaul. Secure search via sketching for homomorphic encryption. In *Proceedings of the 25th ACM Conference on Computer and Communications Security*. ACM, 2018.

[3] Anonymous. ReportLib: Open library for FHE search, with an example system, will be published upon acceptance or reviewer's request., 2018.

[4] Anonymous. Secure database queries on the cloud: Homomorphic encryption meets coresets, full version., 2018.

[5] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private database queries using somewhat homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, pages 102–118. Springer, 2013.

[6] C. Bösch, P. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):18, 2015.

[7] C. Boutsidis, A. Zouzias, and P. Drineas. Random projections for k-means clustering. In *Advances in Neural Information Processing Systems*, pages 298–306, 2010.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[9] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.

[10] V. Braverman, G. Frahling, H. Lang, C. Sohler, and L. F. Yang. Clustering high dimensional dynamic data streams. *arXiv preprint arXiv:1706.03887*, 2017.

[11] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku. Streaming k-means on well-clusterable data. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 26–40. Society for Industrial and Applied Mathematics, 2011.

[12] G. S. Çetin, W. Dai, Y. Doröz, W. J. Martin, and B. Sunar. Blind web search: How far are we from a privacy preserving search engine? *IACR Cryptology ePrint Archive*, 2016:801, 2016.

[13] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 577–594. Springer, 2010.

[14] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. *IACR Cryptology ePrint Archive*, 2017:299, 2017.

[15] J. Chen and Q. Zhang. Bias-aware sketches. *Proceedings of the VLDB Endowment*, 10(9):961–972, 2017.

[16] J. H. Cheon, M. Kim, and M. Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Trans. Information Forensics and Security*, 11(1):188–199, 2016.

[17] J. H. Cheon, M. Kim, and K. E. Lauter. Homomorphic computation of edit distance. In *Financial Cryptography Workshops*, pages 194–212, 2015.

[18] J. H. Cheon, M. Kim, and K. E. Lauter. Homomorphic computation of edit distance. In *Financial Cryptography Workshops*, pages 194–212, 2015.

[19] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords, 1997.

[20] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[21] K. L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010.

[22] E. Cohen and H. Kaplan. Tighter estimation using bottom k sketches. *Proceedings of the VLDB Endowment*, 1(1):213–224, 2008.

[23] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 743–749. ACM, 2005.

[24] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large data bases*, pages 13–24. VLDB Endowment, 2005.

[25] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 281–292. ACM, 2007.

[26] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[27] A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler. $(1+\varepsilon)$-approximation for facility location in data streams. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1710–1728. Society for Industrial and Applied Mathematics, 2013.

[28] C. Ding, D. Pei, and A. Salomaa. *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.

[29] Y. Doröz, B. Sunar, and G. Hammouri. Bandwidth efficient PIR from NTRU. In *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*, pages 195–207, 2014.

[30] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 201–210. JMLR.org, 2016.

[31] D. Du, F. K. Hwang, and F. Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

[32] M. Garofalakis, D. Keren, and V. Samoladas. Sketch-based geometric monitoring of distributed stream queries. *Proceedings of the VLDB Endowment*, 6(10):937–948, 2013.

[33] C. Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009. AAI3382729.

[34] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[35] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, volume 1, pages 79–88, 2001.

[36] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM.

[37] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[38] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[39] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *Proceedings of the 15th International Conference on Information Security and Cryptology*, ICISC'12, pages 1–21, Berlin, Heidelberg, 2013. Springer-Verlag.

[40] S. Guha and A. McGregor. Graph synopses, sketches, and streams: A survey. *Proceedings of the VLDB Endowment*, 5(12):2030–2031, 2012.

[41] S. Halevi. Helib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/, 2013.

[42] S. Halevi and V. Shoup. Algorithms in helib. In *34rd Annual International Cryptology Conference, CRYPTO 2014*. Springer Verlag, 2014.

[43] H. Huang and S. P. Kasiviswanathan. Streaming anomaly detection using randomized matrix sketching. *Proceedings of the VLDB Endowment*, 9(3):192–203, 2015.

[44] J. Huggins, T. Campbell, and T. Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems*, pages 4080–4088, 2016.

[45] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, pages 363–372, 2000.

[46] P. Indyk, H. Q. Ngo, and A. Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1126–1142. SIAM, 2010.

[47] M. Kim, H. T. Lee, S. Ling, S. Q. Ren, B. H. M. Tan, and H. Wang. Better security for queries on encrypted databases. *IACR Cryptology ePrint Archive*, 2016:470, 2016.

[48] M. Kim, H. T. Lee, S. Ling, B. H. M. Tan, and H. Wang. Private compound wildcard queries using fully homomorphic encryption. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[49] K. E. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. *LATINCRYPT*, 8895:3–27, 2014.

[50] K. E. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. *IACR Cryptology ePrint Archive*, 2015:133, 2015.

[51] W. Lu, S. Kawasaki, and J. Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. *IACR Cryptology ePrint Archive*, 2016:1163, 2016.

[52] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.

[53] A. Name. CHELib: Open library for coresets in fhe, 2018.

[54] J. M. Phillips. Coresets and sketches. *arXiv preprint arXiv:1601.00617*, 2016.

[55] G. Reeves, J. Liu, S. Nath, and F. Zhao. Managing massive time series streams with multi-scale compressed trickles. *Proceedings of the VLDB Endowment*, 2(1):97–108, 2009.

[56] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[57] S. S. Roy, F. Vercauteren, J. Vliegen, and I. Verbauwhede. Hardware assisted fully homomorphic function evaluation and encrypted search. *IEEE Transactions on Computers*, 2017.

[58] 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*, pages 44–55. IEEE, 2000.

[59] H. Tang, X. Jiang, X. Wang, S. Wang, H. Sofia, D. Fox, K. Lauter, B. Malin, A. Telenti, L. Xiong, and L. Ohno-Machado. Protecting genomic data analytics in the cloud: state of the art and opportunities. *BMC Medical Genomics*, 9(1):63, Oct 2016.

[60] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical private queries on public data. In *NSDI*, pages 299–313. USENIX Association, 2017.

[61] D. P. Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[62] A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

[63] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop*, CCSW '13, pages 65–76, New York, NY, USA, 2013. ACM.

[64] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. New packing method in somewhat homomorphic encryption and its applications. *Sec. and Commun. Netw.*, 8(13):2194–2213, Sept. 2015.