EPiC: Efficient Privacy-Preserving Counting for MapReduce

***Abstract*—In the face of an untrusted cloud infrastructure, outsourced data needs to be protected. Fully homomorphic encryption is one solution that also allows performing operations on outsourced data. However, the involved high overhead of today's fully homomorphic encryption techniques outweigh cloud cost saving advantages, rendering it impractical. We present EPiC, a practical, efficient protocol for the privacy-preserving evaluation of a fundamental operation on data sets: frequency counting. In an IND-CPA encrypted outsourced data set, a cloud user can specify a pattern, and the cloud will count the number of occurrences of this pattern in a completely oblivious manner. A pattern is expressed as a boolean formula on the fields of the records and can specify values counting, range counting, and conjunctions/disjunctions of field values. EPiC's main idea is, first, to reduce the problem of counting to a summation of polynomial evaluations. Second, to efficiently evaluate the summation of polynomial evaluations in a privacy-preserving manner, we extend previous work on the Hidden Modular Group Order assumption and design a new *somewhat homomorphic* encryption scheme. We show how a general pattern, defined by a boolean formula, is arithmetized into a multivariate polynomial over $GF(2)$ and used in EPiC. This scheme is highly efficient in our particular counting scenario. Besides a formal analysis where we prove EPiC's privacy, we also present implementation and evaluation results. We specifically target Google's prominent MapReduce paradigm as offered by major cloud providers. Our evaluation performed both locally and in Amazon's public cloud with data sets sizes of up to 1 TByte shows only modest overhead compared to non-private counting, attesting to EPiC's efficiency.**

I. Introduction

Cloud computing is a promising technology for larger enterprises and even governmental organizations. Major cloud computing providers such as Amazon or Google offer to outsource their data and computation. The main idea is that a cloud user can not only move his data to the cloud, but also send operations ("algorithms", "programs", "code") on his data to the cloud. The main advantage for users lies in the clouds' flexible cost model: users are only charged by use, e.g., by the total amount of storage or CPU time used. In addition, clouds offer "elastic" services that can scale with the users' demands. For example, during peak times, users can rent additional resources from the cloud. Consequently, instead of maintaining their own data centers, users can save costs by using cloud technologies.

One cloud computing framework that allows outsourcing data and operating on outsourced data is Google's prominent MapReduce API [11]. MapReduce is offered by major public cloud providers today, such as Amazon [1], Google [17], IBM [20] or Microsoft [24]. MapReduce is typically used for analysis operations on huge amounts of (outsourced) data, e.g., scanning through data and finding patterns, counting occurrences of specific patterns, and other statistics [18].

While the idea of moving data and computation to a (public) cloud for cost savings is appealing, many new privacy questions arise if delicate information is outsourced. The main problem is that as soon as a cloud user moves his services (data+computation) to a cloud, he automatically relinquishes control. The user has to trust the cloud to store and protect data against *adversaries*. Examples for adversaries can be hackers that break into the cloud, i.e., the data center, to steal data. Also, insiders such as data center administrative staff can easily access data. As multiple cloud users are hosted on the same data center ("multi tenancy"), even other cloud users might try to illegally access data. Finally, as cloud providers place data centers abroad in foreign countries with unclear privacy laws, local authorities are threatening outsourced data. Such attacks are realistic and have already been reported in the real-world [16, 27, 28, 32, 36]. In conclusion, the cloud cannot be trusted as there are many ways that adversaries might violate data *privacy*.

While the encryption of data is a viable privacy protection mechanism, it renders subsequent operations on encrypted data by the cloud a challenging problem. To address this problem, *fully homomorphic* encryption techniques have recently been investigated, cf. Gentry [13] or see Lipmaa [23] for an overview. Fully homomorphic encryption guarantees that the cloud neither learns details about the stored data nor about the results it computes. The problem with fully homomorphic encryption, however, is its involved enormous complexity and therewith costs. As of today, solutions are inefficient [14, 26], and a deployment in a real-world cloud would outweigh any cost advantage offered by the cloud. Furthermore, any solution running in a real-world cloud needs to be tailored to the specifics of the cloud computing paradigm, e.g., MapReduce. MapReduce comprises a specific two-phase setup, where (first) the workload is parallelized in the Map-phase, and (second) individual results are aggregated during the Reduce-phase to present a combined result to the user.

This paper presents an efficient, practical, yet privacy-preserving protocol for a fundamental data analysis primitive in MapReduce: *counting occurrences* of patterns. In an outsourced data set comprising a large number of various patterns, EPiC, "Efficient PrIvacy-preserving Counting for MapReduce", allows the cloud user to specify a (plaintext) pattern, and the cloud will count the number of occurrences of this pattern in the stored ciphertexts without detecting

which pattern is being counted or how often the pattern occurs. A pattern is expressed as a boolean formula on the fields of the records and can therefore specify a specific field value, a range of field values, but also more complex patterns consisting of conjunctions/disjunctions of fields values. For example, a pattern could refer to $\texttt{age} \in [50, 70] \wedge (diabetes = 1 \vee hypertension = 1)$. This allows, e.g., the oblivious computation of histograms by the cloud. The main idea of EPiC is to transform the problem of privacy-preserving pattern counting into a summation of polynomial evaluations. Our work is inspired by Lauter et al. [22] to use *somewhat homomorphic* encryption to address specific privacy-preserving operations. We show that the summation of polynomial evaluations can be efficiently computed in a privacy-preserving manner using a specific *somewhat homomorphic* encryption scheme. To this effect, we extend previous work on the Hidden Modular Group Order assumption and design a new *somewhat homomorphic* encryption scheme. We also show how a general pattern, defined by a boolean formula, is arithmetized into a multivariate polynomial over $GF(2)$ and optimized for efficiency. The new encryption mechanism is efficient only in the particular context that we target in this paper, the summation of polynomial evaluations with the total number (the "domain") of different patterns being relatively small.

In conclusion, the contributions of this paper are:

- EPiC, a new protocol to enable privacy-preserving pattern counting in MapReduce clouds. EPiC reduces the problem of counting occurrences of a pattern to the summation of the pattern indicator-polynomial evaluations on the cloud encrypted records.
- A new "somewhat homomorphic" IND-CPA encryption scheme that addresses the secure summation of polynomial evaluations for counting in a highly efficient manner.
- An implementation of EPiC and its encryption mechanism together with an evaluation in a realistic setting. The source code is available for download [4].

II. Problem Statement

Overview: We will use an example application to motivate our work. Along the lines of recent reports [33], imagine a hospital scenario where patient records are managed electronically. To reduce cost and grant access to, e.g., other hospitals and external doctors, the hospital refrains from investing into an own, local data center, but plans to outsource patient records to a public cloud. Regulatory matters require the privacy-protection of sensitive medical information, so outsourced data has to be encrypted. However, besides uploading, retrieving or editing patient records performed by multiple entities (hospitals, doctors etc.), one entity eventually wants to collect some statistics on the outsourced patient records without the necessity of downloading all of them.

A. Cloud Counting

More specifically, we assume that each patient record R, besides raw data such as maybe a picture or some doctors' notes, also includes one or more fields $R.c$ containing some patterns. In practice, this field could denote the category or type of disease a patient is suffering from, e.g., "diabetes" or "hypertension". While one (or more) cloud users $\mathcal{U}$ add, remove or edit records, eventually one cloud user $\mathcal{U}$ wants to know, how many patients suffer from disease χ. That is, user $\mathcal{U}$ wants to extract the frequency of occurrence of pattern χ and therewith how many records contain $R.c = \chi$. Due to the large amount of data, downloading each patient record is prohibitive, and the counting should be performed by the cloud.

While encryption of data, access control, and key management in a multi-user cloud environment are clearly important topics, we focus on the problem of a-posteriori extracting information out of the outsourced data in a privacy-preserving manner. The cloud must neither learn details about the data stored, nor any information about the counting, what is counted, the count itself etc. Instead, the cloud processes $\mathcal{U}$'s counting queries "obliviously".

We will now first specify the general setup of counting schemes for public clouds and then formally define privacy requirements. Note that throughout the rest of this paper, we will assume the "pattern" a user $\mathcal{U}$ might look for to be "countable".

Definition 1 (Cloud Counting): Let $\mathcal{R}$ denote a sequence of records $\mathcal{R} := \{R_1, \ldots, R_n\}$. Besides some random data, each record R_i contains up to m different countable fields $R_i.c_k \in \mathcal{D}_k$, where $\mathcal{D}_k$ is the domain of $R_i.c_k$, $1 \leq k \leq m$. Without loss of generality, we assume $\mathcal{D}_k = \{0, 1, \ldots, |\mathcal{D}_k| - 1\}$, $|\mathcal{D}_k|$ denotes size of $\mathcal{D}_k$. A privacy-preserving counting scheme comprises the following probabilistic polynomial time algorithms:

1) KeyGen(s) : using a security parameter s, KeyGen outputs a secret key $\mathcal{S}$.
2) Encrypt$(\mathcal{S}, \mathcal{R})$: uses secret key $\mathcal{S}$ to encrypt the sequence of records $\mathcal{R}$. The output is a sequence of encryptions of records $\mathcal{E} := \{E_{R_1}, \ldots, E_{R_n}\}$, where E_{R_i} denotes the encryption of record R_i.
3) Upload$(\mathcal{E})$: uploads the sequence of encryptions $\mathcal{E}$ to the cloud.
4) PrepareQuery$(\mathcal{S}, \chi)$: this algorithms generates a query Q out of secret $\mathcal{S}$ and the values $\chi = (\chi_1, \ldots, \chi_m)$ to be counted, with $\chi_k \in \mathcal{D}_k$.
5) ProcessQuery$(Q, \mathcal{E})$: performs the actual counting. Uses a query Q, the sequence of ciphertexts $\mathcal{E}$, and outputs a result E_Σ.
6) Decode$(\mathcal{S}, E_\Sigma)$: takes secret $\mathcal{S}$ and query result E_Σ to output a final sum σ, such that

$$\sigma = \sum_{i=1}^{n} f_\chi(R_i.c_1, \ldots, R_i.c_m),$$

if E_Σ = ProcessQuery$(Q, \mathcal{E})$ with Q = PrepareQuery$(\mathcal{S}, \chi)$, $\mathcal{E}$ = Encrypt$(\mathcal{S}, \mathcal{R})$, and

$$f_\chi(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} = \chi \\ 0, & \text{otherwise.} \end{cases}$$

Here, $\mathbf{x} = (x_1 \ldots, x_m), x_k \in \mathcal{D}_k$.

According to this definition, the cloud user $\mathcal{U}$ encrypts the sequence of records and uploads them into the cloud. If $\mathcal{U}$ want to know the number of occurrences of χ in the records, he prepares a query Q, sends Q to the cloud, and the cloud processes Q. Finally, the cloud sends a result E_Σ back to U who can decrypt this result and learn the number of occurrences of χ. The idea of a performing the counting in the cloud is to put the main computational burden on the cloud side. Both storage and computational overhead for KEYGEN, ENCRYT, UPLOAD, PREPAREQUERY, and DECODE should be lightweight compared to PROCESSQUERY.

B. Privacy

In the face of untrusted cloud infrastructure, cloud user $\mathcal{U}$ wants to perform counting in a privacy-preserving manner. Intuitively, the data stored at the cloud as well as the counting operations must be protected against a curious cloud. Informally, we demand 1.) *storage privacy* and 2.) *counting privacy* against the cloud which we will now call "adversary $\mathcal{A}$". This adversary $\mathcal{A}$ should only learn "trivial" privacy properties like the total size of outsourced data, the total number of patient records or the number of counts performed for $\mathcal{U}$.

With *storage privacy*, we capture the intuition that, by storing data and counting, the cloud should not learn any information about the content it stores. In addition, *counting privacy* captures the problem that, again by storing data and counting, the cloud should not learn any details about the counting performed, e.g., which value is counted, whether a value is counted twice or what the resulting count is.

Inspired by traditional indistinguishability [15], we formalize our privacy requirements using a game based definition. Our privacy games for storage privacy (GAME_1) and counting privacy (GAME_2) are played between adversary $\mathcal{A}$ (representing the cloud) and a challenger (representing user $\mathcal{U}$).

Both games comprise a learning and a challenge phase. The learning phase, cf. Algorithm 1, is the same for GAME_1 and for GAME_2. The difference lies only in the challenge phases. While one could certainly join the two different privacy notions and games into one (and we actually prove that storage privacy *implies* counting privacy later in Section IV), we stick to the separated setup for ease of understanding for now.

Algorithm 1: Learning Phase GAME_1 and GAME_2

Challenger:	$\mathcal{S} := \text{KEYGEN}(s)$;
for $i := 1$ **to** T **do**	
$\mathcal{A} \rightarrow$ Challenger:	$\mathcal{R} := \{R_1, \ldots, R_n\}, \chi$;
Challenger:	$\mathcal{E} := \text{ENCRYPT}(\mathcal{S}, \mathcal{R})$;
	$Q := \text{PREPAREQUERY}(\mathcal{S}, \chi)$;
	$E_\Sigma := \text{PROCESSQUERY}(Q, \mathcal{E})$;
	$\sigma := \text{DECODE}(\mathcal{S}, E_\Sigma)$;
Challenger$\rightarrow \mathcal{A}$:	$\{\mathcal{E}, Q, E_\Sigma, \sigma\}$;
end	

Learning Phase GAME_1 *and* GAME_2: First, the challenger executes KEYGEN to derive a new secret key S, and $\mathcal{A}$ enters the learning phase. Here, $\mathcal{A}$ computes a sequence of records $\mathcal{R}$ and a value χ to be counted and sends it to the challenger. The challenger encrypts the sequence of records, and prepares a new query based on the supplied χ. Finally, the challenger counts for χ, i.e., executes the PROCESSQUERY algorithm and sends the encrypted records, the query, and the (encrypted) result back to $\mathcal{A}$. This interaction between $\mathcal{A}$ and the challenger is repeated T times.

Algorithm 2: Challenge Phase $\text{GAME}_1(\mathcal{A})$

$\mathcal{A} \rightarrow$ Challenger:	$(\mathcal{R}_0, \chi^0)$, $(\mathcal{R}_1, \chi^1)$, $\lvert\mathcal{R}_0\rvert = \lvert\mathcal{R}_1\rvert$, $\lvert\chi^0\rvert = \lvert\chi^1\rvert$;
Challenger:	$b \leftarrow \{0, 1\}$;
	$\mathcal{E}_b := \text{ENCRYPT}(\mathcal{S}, \mathcal{R}_b)$;
	$Q_b := \text{PREPAREQUERY}(\mathcal{S}, \chi^b)$;
	$E_{\Sigma_b} := \text{PROCESSQUERY}(Q_b, \mathcal{E}_b)$;
Challenger$\rightarrow \mathcal{A}$:	$\{\mathcal{E}_b, Q_b, E_{\Sigma_b}\}$;
$\mathcal{A}$:	guess b';
if $b' = b$ **then**	
output 1;	

Challenge Phase $\text{GAME}_1(\mathcal{A})$*:* In the challenge phase, cf. Algorithm 2, $\mathcal{A}$ selects a distinct pair of sequences of records and queries $(\mathcal{R}_0, \chi^0)$ and $(\mathcal{R}_1, \chi^1)$ with $|\mathcal{R}_0| = |\mathcal{R}_1|, |\chi^0| = |\chi^1|$ and sends it to the challenger. The challenger randomly selects $b \in \{0, 1\}$ and executes $\mathcal{E}_b := \text{ENCRYPT}(\mathcal{S}, \mathcal{R}_b)$ to encrypt a sequence of records, $Q_b := \text{PREPAREQUERY}(\mathcal{S}, \chi^b)$ to generate a new query Q_b, and $E_{\Sigma_b} := \text{PROCESSQUERY}(Q_b, \mathcal{E}_b)$ to generate a result. All this is sent back to $\mathcal{A}$. Therewith, $\mathcal{A}$ guesses b'. The outcome of GAME_1 is 1, if $b = b'$.

Algorithm 3: Challenge Phase $\text{GAME}_2(\mathcal{A})$

$\mathcal{A} \rightarrow$ Challenger:	$\mathcal{R}, \chi^0, \chi^1$, $\lvert\chi^0\rvert = \lvert\chi^1\rvert$;
Challenger:	$b \leftarrow \{0, 1\}$;
	$\mathcal{E} := \text{ENCRYPT}(\mathcal{S}, \mathcal{R})$;
	$Q_b := \text{PREPAREQUERY}(\mathcal{S}, \chi^b)$;
	$E_{\Sigma_b} := \text{PROCESSQUERY}(Q_b, \mathcal{E})$;
Challenger$\rightarrow \mathcal{A}$:	$\{\mathcal{E}, Q_b, E_{\Sigma_b}\}$;
$\mathcal{A}$:	guess b'
if $b' = b$ **then**	
output 1;	

Challenge Phase $\text{GAME}_2(\mathcal{A})$*:* $\mathcal{A}$ selects a sequence of records $\mathcal{R}$, two distinct values to be counted $\{\chi^0, \chi^1\}$, and sends $\{\mathcal{R}, \chi^0, \chi^1\}$ to the challenger. Now, the challenger randomly selects $b \in \{0, 1\}$ and executes $\mathcal{E} := \text{ENCRYPT}(\mathcal{S}, \mathcal{R})$ to encrypt the sequence of records, $Q_b := \text{PREPAREQUERY}(\mathcal{S}, \chi^b)$ to generate a new query Q_b, and finally $E_{\Sigma_b} := \text{PROCESSQUERY}(Q_b, \mathcal{E})$. The challenger sends $\{\mathcal{E}, Q_b, E_{\Sigma_b}\}$ back to $\mathcal{A}$. $\mathcal{A}$ guesses b'. The outcome of GAME_2 is 1, if $b = b'$.

Privacy Definition After the description of GAME_1 and GAME_1, we can now formally define privacy for cloud-based

counting.

Definition 2 (Privacy): A cloud counting scheme is $(T, \epsilon_1, \epsilon_2)$-privacy-preserving, *iff*

$$Pr(\text{GAME}_1(\mathcal{A}) = 1) \leq \frac{1}{2} + \epsilon_1(s) \text{ and}$$
$$Pr(\text{GAME}_2(\mathcal{A}) = 1) \leq \frac{1}{2} + \epsilon_2(s)$$

for all probabilistic polynomial time adversaries $\mathcal{A}$ with running time T. Functions $\epsilon_1(s)$ and $\epsilon_2(s)$ are negligible, i.e., $\epsilon_1(s) < s^{-n_1}$ and $\epsilon_2(s) < s^{-n_2}$ for any $n_1, n_2 \in \mathbb{N}$ and sufficiently large security parameter s.

Discussion: The difference between GAME_1 and GAME_2 is $\mathcal{A}$'s goal. In the real-world, GAME_1 reflects an adversary who knows or can even manipulate the data to be stored and the queries made, and he sees query results. $\mathcal{A}$'s goals is to learn something (new) about the *data* stored in the cloud. If Definition 2 holds, than any two "transcripts" that $\mathcal{A}$ sees, i.e., two sets of encrypted data, queries, and encrypted query results, are computationally indistinguishable for $\mathcal{A}$.

In GAME_2, $\mathcal{A}$'s goal is, by using the same means as in GAME_1, to learn something (new) about the *query* (and their results). Here, any two "transcripts" of queries and encrypted query results are computationally indistinguishable for $\mathcal{A}$.

In the real-world, such an adversary that we envision could be the cloud infrastructure.

Limitations: Note that the adversary must specify the same length for the two sets of patient records in GAME_1. Otherwise, just by looking at the size of the encrypted records, $\mathcal{A}$ could win GAME_1. While there exist mitigation strategies, e.g., by using padding and artificially increasing the size of the data, these are typically contradictory to cloud efficiency and low cost. There are also other, "trivial" privacy properties that can be leaked in a scenario like ours. For example, the fact that a doctor makes a certain number of queries, or queries at certain times during the day might leak some information about the outsourced data. Again, mitigation strategies exists, e.g., fake queries, but we leave this for future work. We conjecture that loss of those "trivial" privacy properties can be acceptable in many real-world scenarios. We also consider only *semi-honest* clouds ("honest-but-curious") in this paper. A fully malicious cloud could selectively carry out a DoS-attack, deviate from protocol execution and try to perform attacks similar to "reaction attacks" [19]. While these attack are certainly valid, we leave them for future work.

C. MapReduce

The efficiency of counting relies on the performance of PROCESSQUERY which involves processing huge data in the cloud. Cloud computing usually processes data in parallel via multiple nodes in the cloud data center based on some computation paradigm. For efficiency, PROCESSQUERY has to take the specifics of that computation into account. One of the most widespread, frequently used framework for distributed computation that is offered by major cloud providers today is MapReduce. In the following, we will give a very compressed overview about MapReduce, only to understand EPiC. For more details, refer to Dean and Ghemawat [11].

In the MapReduce framework, a user uploads his data into the cloud. During upload, data is automatically split into pieces (*InputSplits*) and distributed among the nodes in the cloud's data center. If the user wants the cloud to perform an operation on the outsourced data, he uploads an implementation of his operation, e.g., Java .class files, to the cloud. More precisely, the user has to provide implementations of two functions, the so called *map* function and the *reduce* function – these two functions will be executed by the cloud on the user's data.

A MapReduce "job" runs in two phases. Nodes in the data center storing an InputSplit (*Mapper* nodes) scan through their InputSplit and evaluate the user's map function on data. This operation is performed by all Mappers in parallel. The output of each Map function evaluation is a set of key-value pairs. All key values pairs are sent to so called *Reducer* nodes.

The Reducer nodes collect key-value pairs emitted by Mappers and aggregate them using the user provided reduce function. Reducers produce a final output that is sent back to the user.

This setup takes advantage of the parallel nature of a cloud data center and allows for scalability and elasticity.

III. EPIC PROTOCOL

Before presenting EPiC, we briefly discuss why possible straightforward, **trivial solutions do not work** in our particular application scenario. This motivates the need for more sophisticated solutions such as EPiC.

Precomputed Counters: One could imagine that the cloud user, in the purpose of counting a value χ_k in a single countable field $\mathcal{D}_k$, simply stores encrypted counters for each possible value of χ_k in domain $\mathcal{D}_k$ in the cloud. Each time records are added, removed or updated, the cloud user updates the encrypted counters.

However, this approach does not scale very well in our scenario where multiple cloud users (different "doctors") perform updates and add or modify records. An expensive user side locking mechanism would be required to ensure consistency of the encrypted counter values. Moreover, in the case of more complex queries involving multiple countable fields, all possible combinations of counters would need to be updated by users involving a lot of user side computation.

Per-Record Counters ("Voting"): Alternatively and similar to a naive voting scheme, to enable counting for a single countable field $\mathcal{D}_k$, each encrypted record stored in the cloud could be augmented with an encrypted bit field of length $\log_2 n \cdot |\mathcal{D}_k|$ bits. Each subset of $\log_2 n$ bits represents one of the values of $\mathcal{D}_k$. If a record's countable value in field $\mathcal{D}_k$ matches the value corresponding to a subset, then the LSB of the according subset is set to 1. The cloud only sums the encrypted bit fields (using additive homomorphic encryption) for all records.

Again, such an approach does not allow flexible queries. to support counting on m fields, the user would need to compute 2^m counters (2^m combinations for m fields). This puts a

high burden on the user in case of adding or removing fields. Furthermore, in case of m fields, a modifying a record on any field needs m updates to 2^{m-1} counters related to that field. So, updating a record not only requires updating countable fields, but also requires updating 2^{m-1} counters related to those fields.

In conclusion, straightforward solutions to address our particular problem setting do not provide an efficient, practical, and flexible counting solution for multi-user, multiple field data sets.

A. EPiC Overview

For ease of exposition, we first introduce EPiC for the simpler case of counting on only one field $\mathcal{D}$. Later, in Section III-E, we will extend EPiC to support counting on boolean combinations of multiple fields $\mathcal{D}_1, \ldots, \mathcal{D}_m$.

EPiC's main rationale is to perform the counting in the cloud by evaluating a polynomial. As query Q, cloud user $\mathcal{U}$ sends an *indicator polynomial* $P_\chi(x)$ to the cloud that is specific to the value χ he is interested in. Conceptually, the cloud evaluates $P_\chi(x)$ on each stored record's countable value $R_i.c$. The outcome of all individual polynomial evaluations is a (large) set of values of either "1" or "0". The cloud now simply adds these values and sends the sum back to $\mathcal{U}$. Based on the received sum of evaluated polynomials, $\mathcal{U}$ learns the number of occurrences of χ in the investigated set of records. However, this conceptual approach is not feasible with the somewhat homomorphic encryption available to us. Below, we show how the summation of polynomial evaluations is indirectly computed by first summing the monomials first and then combining them.

There are three challenges with this approach.

First, the cloud has to evaluate indicator polynomial $P_\chi(x)$ (monomials to be precise) on encrypted data. To avoid that the cloud learns any information about the stored data (as in *storage privacy*, see Section II-B), all patient records are encrypted using an IND-CPA encryption mechanism.

Second, although $\mathcal{U}$ sends $P_\chi(x)$ to the cloud for evaluation, the cloud must neither learn χ nor any other information about the query, see *counting privacy*. The query Q, i.e., $P_\chi(x)$ itself must be IND-CPA encrypted, but still its evaluation by the cloud has to be possible.

Third, the polynomial evaluation has to be extremely efficient, as the cloud evaluates $P_\chi(x)$ on each of the potentially huge number n of records. An expensive polynomial evaluation, e.g., based on fully homomorphic encryption, bilinear pairings, modular exponentiations or other expensive cryptographic primitives, would outweigh cloud cost advantages.

Somewhat homomorphic encryption: EPiC addresses the above challenges by employing a new "somewhat" homomorphic encryption scheme that is inspired by the idea of Lauter et al. [22]. Our scheme is originally additive, but we can extend it to allow for multiplication, too, by allowing the size of the ciphertext to grow linearly with the number of multiplications. As EPiC's polynomial evaluation requires only few multiplications, the overhead remains modest – see Section V. Compared to other additive homomorphic schemes such as Paillier's, our encryption has the advantage of using only *integer* addition and multiplication instead of expensive modular exponentiation. Moreover, it is far from straightforward how other additive homomorphic schemes such as Paillier's can be converted into an efficient, secure "somewhat" homomorphic encryption scheme. We prove the security of EPiC based on the Hidden Modular Group Order assumption [34].

While EPiC encrypts the countable fields of each record using the new somewhat homomorphic encryption "ENC" to give semantically secure (IND-CPA) encryption, the remainder of the patient record is encrypted using AES-CBC. As we use random IVs, this encryption is also IND-CPA [7].

MapReduce: Moreover, regarding efficiency, we stress the fact that EPiC's setup using polynomial evaluation seamlessly suits the MapReduce paradigm: the large amount of data, patient records, can be split and distributed for homomorphic polynomial evaluation in parallel by different Mappers in the cloud's data center. The aggregation, i.e., the homomorphic addition of encrypted values can finally be done by Reducers within the reduce phase.

B. Polynomial Counting

If $\mathcal{U}$ wishes to count occurrences of χ in the countable field $R_i.c \in \mathcal{D}$, the idea is that $\mathcal{U}$ prepares an *indicator polynomial*:

$$P_\chi(x) = \begin{cases} 1, & \text{if } x = \chi \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

and asks the cloud to scan through the set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of all patient records to compute the sum $\sum_x P_\chi(x)$. The result is the number of occurrences of χ in the data set.

One way to generate a polynomial $P_\chi(x)$ is to construct the interpolation polynomial in the Lagrange form:

$$P_\chi(x) := \sum_{j=0}^{|\mathcal{D}|-1} a_j \cdot x^j := \prod_{x_j \neq \chi} \frac{x - x_j}{\chi - x_j}, \tag{2}$$

where x_j are all possible elements of $\mathcal{D}$ except χ. The polynomial $P_\chi(x)$ is of degree $|\mathcal{D}| - 1$, and its coefficients a_j are uniquely determined based on χ.

In EPiC, the countable fields $R_i.c$ are IND-CPA encrypted to E_{c_i} using the somewhat homomorphic encryption. User $\mathcal{U}$ encrypts coefficients a_j to E_{a_j} and sends them to the cloud, which now computes the encrypted sum

$$\begin{aligned} E_\Sigma &:= \sum_{i=1}^{n} P_\chi(E_{c_i}) \\ &= \sum_{i=1}^{n} \sum_{j=0}^{|\mathcal{D}|-1} E_{a_j} \cdot (E_{c_i})^j \\ &= \sum_{j=0}^{|\mathcal{D}|-1} (E_{a_j} \cdot \sum_{i=1}^{n} (E_{c_i})^j). \end{aligned} \tag{3}$$

User $\mathcal{U}$ receives back the result E_Σ and can now decrypt the count of χ

$$\sigma := \textsc{Dec}(E_\Sigma) = P_\chi(x).$$

As you can see in Equation (3), this (encrypted) polynomial evaluation is a two step process: *first*, the cloud computes for each c_i all $|\mathcal{D}| - 1$ possible monomials and sums up monomials with the same exponent. *Second*, the cloud multiplies the summed monomials with their coefficients and sums up results.

Note that, if $|\mathcal{D}|$ is reasonably small, the sums $\sum_{i=1}^{n} (E_{c_i})^j$ can be evaluated efficiently by the cloud, see Section V.

C. Somewhat homomorphic encryption

We now describe EPiC's somewhat homomorphic encryption scheme using standard notation. Note that our scheme is a secret key homomorphic encryption scheme.

Main Idea: The polynomial counting technique requires the cloud to be able to compute, for each record, P_χ on the countable field and add results to finally return the sum to the user. This method assumes homomorphic properties of the encryption scheme. The main idea of our encryption scheme is borrowed from the Hidden Modular Group Order assumption scheme by Trostle and Parrish [34]. To encrypt a plaintext $\mathcal{P}$, a random number r is selected and, together with some system parameter q, added to $\mathcal{P}$, i.e., $r \cdot q + \mathcal{P}$. Finally, a random parameter $b \in \mathbb{Z}_p$ for some large prime p is multiplied, resulting in ciphertext $\mathcal{C} := b \cdot (r \cdot q + \mathcal{P}) \bmod p$. This setup has the interesting property that the random $r \cdot q$ can be "cancelled out" during decryption by computing $\bmod q$. However before, b is removed by multiplying with $b^{-1} \bmod p$. Informally speaking, the Hidden Modular Group Order assumption by Trostle and Parrish [34] now states that if p and b are sufficiently large and secret, an adversary cannot compute $\mathcal{P}$. We formally prove IND-CPA for this encryption later in Section IV.

We choose this special somewhat homomorphic scheme simply due to its efficiency for the most crucial operation: the per record computation of polynomials in the cloud. As the cloud has to scan through a large amount of records, it needs to evaluate a huge number of polynomials. The way we perform the exponentiation for monomials is around two orders of magnitude faster than the one by Lauter et al. [22].

Again, we stress that our scheme is not a "general" homomorphic encryption scheme, because only ciphertexts of the same exponents can be added, and the size of the results increases (linearly) with the number of additions and multiplications. This renders it useful only for the specific application that we consider where the domain of the countable field is relatively small. Moreover for sound decryption, our scheme requires to know the total number of multiplications performed. This information, too, is only available in our special scenario.

Detailed Description: EPiC's somewhat homomorphic encryption is defined by the following set of operations.

- $\textsc{KeyGen}(s_1, s_2, n, |\mathcal{D}|)$: Parameters $s_1, s_2 \in \mathbb{N}$ are security parameters, $n \in \mathbb{N}$ represents the upper bound for the total number of records in the data set, and $|\mathcal{D}|$ is the domain size of the countable field. KeyGen computes:
  1) a random prime q such that $q > n$.
  2) a random prime p, where
  $$|p| \geq s_1 + \log_2 n + |q| + (s_2 + |q|)(|\mathcal{D}| - 1).$$
  3) a random $b \in \mathbb{Z}_p$.

  We explain the selection of q and p further below.
  The secret key, the output of KeyGen, is defined as $\mathcal{S} := \{p, b\}$.
- $\textsc{Enc}(\mathcal{P})$: Select random number $r, |r| = s_2$. The plaintext $\mathcal{P}$ is encrypted to ciphertext
  $$\mathcal{C} := b \cdot (r \cdot q + \mathcal{P}) \bmod p.$$
- $\textsc{Dec}(\mathcal{C}^j, j)$: To decrypt, compute
  $$\mathcal{P} = \textsc{Dec}(\mathcal{C}^j, j) := b^{-j} \cdot \mathcal{C}^j \bmod p \bmod q.$$

Note the soundness of our scheme:

$$\begin{aligned} \textsc{Dec}(\mathcal{C}^j, j) &= b^{-j} \cdot \mathcal{C}^j \bmod p \bmod q \\ &= b^{-j} \cdot [b \cdot (r \cdot q + \mathcal{P})]^j \bmod p \bmod q \\ &= (r \cdot q + \mathcal{P})^j \bmod q \\ &= \mathcal{P}^j. \end{aligned}$$

We assume that the decryption mechanism "knows" the exponent j of the ciphertext. This is a valid assumption in our special scenario (as we see later) where the cloud returns the result as a ciphertext containing an exponent $|\mathcal{D}|$ of b. So, $\mathcal{U}$ can decrypt the result by letting $j = |\mathcal{D}|$.

Somewhat Homomorphic Properties: The addition and multiplication operations in our encryption scheme take place in the integers – there is no modulo reduction. In our encryption scheme, the degree of b in a ciphertext $\mathcal{C}$ must be known for the decryption of $\mathcal{C}$ to succeed. Therefore, our scheme provides somewhat homomorphic properties. Specifically, while multiplication of any two ciphertexts is allowed, only addition of ciphertexts that contain the same exponent of b is allowed.

- Additively homomorphic:
  $\textsc{Dec}(\mathcal{C}_1^j + \mathcal{C}_2^j, j)$
  $$\begin{aligned} := \;& b^{-j} \cdot \big[[b \cdot (r_1 \cdot q + \mathcal{P}_1) \bmod p]^j \\ & + [b \cdot (r_2 \cdot q + \mathcal{P}_2) \bmod p]^j\big] \bmod p \bmod q \\ = \;& b^{-j} \cdot [b \cdot (r_1 \cdot q + \mathcal{P}_1) \bmod p]^j \bmod p \bmod q \\ & + b^{-j} \cdot [b \cdot (r_2 \cdot q + \mathcal{P}_2) \bmod p]^j \bmod p \bmod q \\ = \;& \textsc{Dec}(\mathcal{C}_1^j, j) + \textsc{Dec}(\mathcal{C}_2^j, j) \end{aligned}$$
- Multiplicatively homomorphic:
  $\textsc{Dec}(\mathcal{C}_1^j \cdot \mathcal{C}_2^k, j + k)$
  $$\begin{aligned} := \;& b^{-(j+k)} \cdot \big[[b \cdot (r_1 \cdot q + \mathcal{P}_1) \bmod p]^j \\ & \cdot [b \cdot (r_2 \cdot q + \mathcal{P}_2) \bmod p]^k\big] \bmod p \bmod q \\ = \;& b^{-j} \cdot [b \cdot (r_1 \cdot q + \mathcal{P}_1) \bmod p]^j \bmod p \bmod q \\ & \cdot b^{-k} \cdot [b \cdot (r_2 \cdot q + \mathcal{P}_2) \bmod p]^k \bmod p \bmod q \\ = \;& \textsc{Dec}(\mathcal{C}_1^j, j) \cdot \textsc{Dec}(\mathcal{C}_2^k, k) \end{aligned}$$

Ciphertext size: As the multiplication and addition are performed on the integers without modulo operations, the size (measured in number of bits representing the ciphertext) of the multiplication will increase to

$$|\mathcal{C}_1 \cdot \mathcal{C}_2| = |\mathcal{C}_1| + |\mathcal{C}_2| - 1.$$

Accordingly, the size of an exponentiation increases to

$$|\mathcal{C}^j| = j \cdot |\mathcal{C}| - (j-1),$$

and the size of a scalar multiplication increases to

$$|n \cdot \mathcal{C}| = |n| + |\mathcal{C}| = \log_2 n + |\mathcal{C}|.$$

The sum of two ciphertexts, however, increases by at most 1 bit:

$$|\mathcal{C}_1 + \mathcal{C}_2| = \begin{cases} |\mathcal{C}_1| + 1, & \text{if } |\mathcal{C}_1| = |\mathcal{C}_2|. \\ \max(|\mathcal{C}_1|, |\mathcal{C}_2|), & \text{otherwise.} \end{cases}$$

Encrypting Coefficients: Recall that, as in Equation (3), the cloud in a first step, computes different exponents j of the encrypted countable fields E_{c_i} and adds the same exponents together. In the second step, using the encrypted coefficients E_{a_j} received from $\mathcal{U}$, the cloud evaluates the polynomial by multiplying the coefficients with the corresponding sums and adds them together. While the encryption of the countable field is simply $E_{c_i} := \text{ENC}(R_i.c)$, the encryption of the coefficients E_{a_j} must allow the additively homomorphic property among different monomials. That is, the degree of b in ciphertexts $E_{a_j} \sum_{i=1}^{n} (E_{c_i})^j$ must be the same for all j – otherwise addition is impossible.

In our protocol, we encrypt a_j such that E_{a_j} contains b with degree $|\mathcal{D}| - j$. More precisely,

$$E_{a_j} := \text{ENC}(a_j) \cdot (\text{ENC}(1))^{|\mathcal{D}|-j-1} \bmod p \tag{4}$$

Therewith, all monomials have the same degree of b, thus allowing for successful decryption of E_Σ.

Note: although E_{a_j} contains b of degree $|\mathcal{D}| - j$, the size of E_{a_j} is only $|E_{a_j}| \leq |p|$ due to the modulo p operation.

Domain of a_j: During the Lagrange interpolation for coefficients a_j of polynomial $P_\chi(x)$, we need to compute inverse elements. In order to compute inverses, we set $a_j \in GF(q)$. Consequently, $|a_j| \leq |q|$.

Security Parameters: User $\mathcal{U}$ receives E_Σ from the cloud and decrypts it to obtain σ. To enable sound decryption of E_Σ, we need $\sigma \in GF(q)$ and $\sum_{j=0}^{|\mathcal{D}|-1} a_j \cdot \sum_{i=1}^{n} (r_i \cdot q + R_i.c)^j \in \mathbb{Z}_p$. Therewith, for all $R_i.c, j \in \mathcal{D},\ r_i < 2^{s_2}$

$$\begin{aligned} q &> \sigma \\ p &> \sum_{j=0}^{|\mathcal{D}|-1} a_j \cdot \sum_{i=1}^{n} (r_i \cdot q + R_i.c)^j. \end{aligned} \tag{5}$$

Selecting q: Since the maximum count possible is equal to the total number of records in the data set n, parameter q can be chosen as a prime larger than n, but smaller than p:

$$p > q > n. \tag{6}$$

Selecting p: As $a_j \leq q$, $r_i \leq 2^{s_2} - 1$, $R_i.c \leq |\mathcal{D}| - 1$, Equation (5) is rewritten to

$$\begin{aligned} p &> \sum_{j=0}^{|\mathcal{D}|-1} q \cdot \sum_{i=1}^{n} ((2^{s_2}-1) \cdot q + |\mathcal{D}| - 1)^j \\ &= n \cdot q \cdot \sum_{j=0}^{|\mathcal{D}|-1} ((2^{s_2}-1) \cdot q + |\mathcal{D}| - 1)^j \\ &= n \cdot q \cdot \frac{((2^{s_2}-1) \cdot q + |\mathcal{D}| - 1)^{|\mathcal{D}|} - 1}{(2^{s_2}-1) \cdot q + |\mathcal{D}| - 2}. \end{aligned} \tag{7}$$

Since $|\mathcal{D}|$ is much smaller than q, (7) is simplified to

$$p > n \cdot q \cdot (2^{s_2} \cdot q)^{|\mathcal{D}|-1}.$$

Therewith, the size of p must be at least

$$|p| \geq \log_2 n + |q| + (s_2 + |q|) \cdot (|\mathcal{D}| - 1).$$

Thus, the size of p is proportional to the domain size of the countable field.

In addition, as our scheme relies on the Hidden Modular Group Order assumption, a security parameter s_1 has to be added to the size of p, see Trostle and Parrish [34] for more details. Finally, p is a prime of size

$$|p| \geq s_1 + \log_2 n + |q| + (s_2 + |q|) \cdot (|\mathcal{D}| - 1). \tag{8}$$

We will formally prove IND-CPA for this encryption scheme in Section IV.

D. Detailed Protocol Description

We use the notation as introduced in Section II-A.

1) KEYGEN(s)*:* Based on security parameter s, cloud user $\mathcal{U}$ chooses s_1, s_2 for the somewhat homomorphic encryption together with a symmetric key K for a block cipher such as AES. $\mathcal{U}$ also computes KEYGEN($s_1, s_2, n, |\mathcal{D}|$) for the somewhat homomorphic encryption, determining an upper bound n for the total number of patient records that might be stored and a value for the domain $\mathcal{D}$ of the countable field. The secret key $\mathcal{S}$ is the output of the somewhat homomorphic encryption KEYGEN and K, i.e., $\mathcal{S} := \{p, b, K\}$.

2) ENCRYPT($\mathcal{S}, \mathcal{R}$)*:* Assume $\mathcal{U}$ wants to store n patient records $\mathcal{R} = \{R_1, \ldots, R_n\}$. Each record R_i is encrypted separating the countable field $R_i.c$ from the rest of the record.

- $R_i.c$ is encrypted using the somewhat homomorphic encryption mechanism, i.e., $E_{c_i} := \text{ENC}(\{p, b\}, R_i.c)$.
- For the rest of the record R_i, a random initialization vector IV is chosen and the record is $\text{AES}_K - \text{CBC}$ encrypted. Using the random IV makes this encryption IND-CPA.

In conclusion, a record R_i encrypts to

$$E_{R_i} := \{E_{c_i}, IV, \text{AES}_K - \text{CBC}(R_i)\}.$$

The output of ENCRYPT is the sequence of encrypted records. $\mathcal{E} := \{E_{R_1}, \ldots, E_{R_n}\}$.

3) UPLOAD($\mathcal{E}$)*:* Upload simply sends all records as one large file to the MapReduce cloud where the file is automatically split into *InputSplits*.

4) PREPAREQUERY$(\mathcal{S}, \chi)$*:* To prepare a query for χ, $\mathcal{U}$ computes the $|\mathcal{D}|$ coefficients a_j of polynomial $P_\chi(x)$ as described in Section III-B. The coefficients a_j are encrypted according to Equation (4), i.e., $E_{a_j} := \text{ENC}(a_j) \cdot (\text{ENC}(1))^{|\mathcal{D}|-j-1} \bmod p$, and sent to the cloud. The cloud will be using these coefficients to perform the evaluation of $P_\chi(x)$. Consequently in EPiC, the output Q of PREPAREQUERY that is sent to the cloud is

$$Q := \{E_{a_0}, E_{a_1}, \ldots, E_{a_{|\mathcal{D}|-1}}\}.$$

Algorithm 4: PROCESSQUERY

MapReduce Framework:

read E_{a_j}, $j = 0, \ldots, |\mathcal{D}| - 1$ **from** $\mathcal{U}$
select M Mappers and 1 Reducer

For each Mapper M:

init $s_j := 0$, $j = 0, \ldots, |\mathcal{D}| - 1$
forall R_i **in** *InputSplit*(M) **do**
  read $\{E_{c_i}, IV, \text{AES}_K - \text{CBC}(R_i)\}$
  for $j = 0$ **to** $|\mathcal{D}| - 1$ **do**
    $s_j := s_j + E_{c_i}^j$
  end
end
emit $\{j, s_j\}$, $j = 0, \ldots, |\mathcal{D}| - 1$

Reducer R:

init $E_\Sigma := 0$, $S_j := 0$, $j = 0, \ldots, |\mathcal{D}| - 1$
forall $\{j, s_j\}$ **in** *MappersOutput* **do**
  $S_j := S_j + s_j$
end
for $j = 0$ **to** $|\mathcal{D}| - 1$ **do**
  $E_\Sigma := E_\Sigma + E_{a_j} \cdot S_j$
end
write $\{E_\Sigma\}$

5) PROCESSQUERY$(Q, \mathcal{E})$*:* Based on the data set size and the cloud configuration, the MapReduce framework has selected M Mapper nodes and 1 Reducer node. Each Mapper node stores one InputSplit.

Algorithm 4 depicts the specification of EPiC's map and reduce functions that will be executed by the cloud. In the mapping phase, for each input record in their locally stored InputSplits, the Mappers compute in parallel all exponents (from 0 to $|\mathcal{D}| - 1$) of the countable field and add the same exponents together. After the Mappers finish scanning over all records in their InputSplits, the sums of exponents are output as key-value pairs. These pairs contain the exponent j as key, and the computed sum s_j as value. In MapReduce, output of the Mappers is then automatically sent to the Reducer ("emit"). The sums s_j emitted by each Mapper are taken over records in the InputSplit corresponding to that Mapper only. The Reducer, therefore, based on the sums received from all Mappers, combines them together to obtain the *global* sums, i.e., the sums over all records in the data set. In a last step, the Reducer uses the coefficients received from $\mathcal{U}$ to evaluate the polynomial by computing the inner product with the global sums. The result is finally sent back to the cloud user and can be decrypted to obtain the count value.

6) DECODE$(\mathcal{S}, E_\Sigma)$*:* Cloud user $\mathcal{U}$ receives E_Σ and computes $\sigma := \text{DEC}(E_\Sigma, |\mathcal{D}|)$ to obtain.

E. Counting patterns defined by a boolean formula

As shown above, the indicator polynomial can be used to obtain the occurrences of an encrypted value in a single field $R_i.c$. We now extend this technique towards a general solution for counting patterns defined by any boolean combination of *multiple* fields in the data set. That is, each (patient) record can contain multiple countable fields, such as $R_i.c_1, R_i.c_2, \ldots, R_i.c_m$. First, we present EPiC's construction of the indicator polynomial for *conjunctive* combinations of fields. Image $\mathcal{U}$ being interested in counting the number of records where $R_i.c_1 = \chi_1$ **and** $R_i.c_2 = \chi_2$ etc. Besides supporting conjunctive combinations, we extend EPiC to *disjunctive* combinations. Therewith, we generalize EPiC for any boolean combination of the countable fields.

Finally, for improved performance, we demonstrate that countable fields in the data set can be organized as a set of separate *binary* fields. We note that our technique is similar to the arithmetization technique, used in Babai and Fortnow [6], Shamir [30].

Conjunctive counting: Assume cloud user $\mathcal{U}$ is interested in counting the number of records that have their countable fields set to the pattern $\chi = (\chi_1, \ldots, \chi_m)$. Here, χ_k, $1 \leq k \leq m$, denotes the value of the k-th field. Let $\varphi = (x_1 = \chi_1 \wedge \ldots \wedge x_m = \chi_m)$ be the conjunction among m fields in the data set. User $\mathcal{U}$ can now construct

$$P_\varphi(\mathbf{x}) = \prod_{k=1}^{m} P_{\chi_k}(x_k), \tag{9}$$

where $\mathbf{x} = (x_1, \ldots, x_m)$ denotes the variables in the multivariate polynomial $P_\varphi(\mathbf{x})$, and $P_{\chi_k}(x_k)$ is the indicator polynomial of the k-th field at value χ_k as before. Therewith, $P_\varphi(\mathbf{x})$ is the indicator polynomial for the desired pattern χ.

Denote $\mathcal{D}_k$ as domain of the k-th field. The domain of $\mathbf{x}$ is $\mathcal{D} := \mathcal{D}_1 \times \cdots \times \mathcal{D}_m$, and the degree of $P_\varphi(\mathbf{x})$ is $\sum_{k=1}^{m} (|\mathcal{D}_k| - 1)$.

$\mathcal{U}$ prepares all coefficients of the indicator polynomial and sends to the cloud as follows. Each coefficient a_J of $P_\varphi(\mathbf{x})$ is uniquely determined by $a_J = a_{1,j_1} \cdot a_{2,j_2} \cdots a_{m,j_m}$, where $J = (j_1, j_2, \ldots, j_m)$, $0 \leq j_k \leq |\mathcal{D}_k| - 1$, and a_J corresponds to the multivariate monomial $x_1^{j_1} \cdot x_2^{j_2} \cdots x_m^{j_m}$ of degree $\sum_{k=1}^{m} j_k$. Algorithm 5 shows the counting process performed by the cloud for conjunctive counting. The computation of monomials and evaluation of the sums of monomials across the records are like those for single field counting (Algorithm 4).

In order for the cloud to evaluate the indicator polynomial $P_\varphi(\mathbf{x})$ over the sums of monomials, the multiplication between encrypted coefficient and the corresponding sum of monomials must result in a ciphertext containing the same exponent of

b. Since each J-th monomial has degree $\sum_{k=1}^{m} j_k$, $\mathcal{U}$ has to encrypt a_j as

$$E_{a_J} := \text{ENC}(1)^{d(J)} \prod_{k=1}^{m} \text{ENC}(a_{k,j_k}),$$

with $d(J) = \sum_{k=1}^{m} (|\mathcal{D}_k| - j_k - 1)$.

After receiving the encrypted evaluated sum E_Σ, $\mathcal{U}$ decrypts E_Σ by using $d = 1 + \sum_{k=1}^{m} (|\mathcal{D}_k| - 1)$ as the decryption degree, i.e., $\sigma = \text{DEC}(E_\Sigma, d)$.

Algorithm 5: PROCESSQUERY – Multiple Counting

MapReduce Framework:

read E_{a_J}, $\forall J \in \mathcal{D} := \mathcal{D}_1 \times \cdots \times \mathcal{D}_m$ **from** $\mathcal{U}$
select M Mappers and 1 Reducer

For each Mapper M:

init $s_J := 0$, $J \in \mathcal{D}$
forall R_i **in** *InputSplit*(M) **do**
  read $\{E_{c_i,1}, \ldots, E_{c_i,m}, IV, \text{AES}_K - \text{CBC}(R_i)\}$
  forall $J := (j_1, \ldots, j_m)$ **in** $\mathcal{D}$ **do**
    $s_J := s_J + \prod_{k=1}^{m} E_{c_{i,k}}^{j_k}$
  end
end
emit $\{J, s_J\}$, $J \in \mathcal{D}$

Reducer R:

init $E_\Sigma := 0$, $S_J := 0$, $J \in \mathcal{D}$
forall $\{J, s_J\}$ **in** *MappersOutput* **do**
  $S_J := S_J + s_J$
end
forall J **in** $\mathcal{D}$ **do**
  $E_\Sigma := E_\Sigma + E_{a_J} \cdot S_J$
end
write $\{E_\Sigma\}$

Prime p is chosen based on the condition

$$|p| \geq s_1 + \log_2 n + |q| + \sum_{k=1}^{m} (s_2 + |q|) \cdot (|\mathcal{D}_k| - 1). \quad (10)$$

The requirement for the prime q, however, remains the same as in (6), because q does not depend on the number and domains of the countable fields.

Disjunctive Counting: Assume the data set contains 2 countable fields $R_i.c_1$ and $R_i.c_2$ among other fields. $\mathcal{U}$'s objective is to count the number of records that have value α in field $R_i.c_1$ **or** value β in field $R_i.c_2$. Let S_α and S_β be the subsets of records that contain only value α in field $R_i.c_1$ and only value β in field $R_i.c_2$, respectively. The desired subset of records that meet the objective is $S_{\alpha\vee\beta} = S_\alpha \cup \mathcal{S}_\beta$. The *size* of $S_{\alpha\vee\beta}$ yields the count value:

$$|S_{\alpha\vee\beta}| = |S_\alpha| + |S_\beta| - |S_{\alpha\wedge\beta}|,$$

where $S_{\alpha\wedge\beta}$ is the subset of records having both α and β in the two fields. The indicator polynomial for this disjunction is, consequently, constructed by

$$P_{\alpha\vee\beta}(\mathbf{x}) := P_\alpha(\mathbf{x}) + P_\beta(\mathbf{x}) - P_{\alpha\wedge\beta}(\mathbf{x}). \quad (11)$$

Supporting conjunctive and disjunctive queries, allows to easily query any boolean expression among the fields. Accordingly, the coefficients of the desired indicator polynomial are uniquely determinable. . Therefore, the cloud computation process, shown in Algorithm 5, is not only useful for conjunctive counting, but also applicable for counting any boolean combinations.

Range Counting: Recall the previous example of two fields $R_i.c_1$ and $R_i.c_2$. If $R_i.c_1$ and $R_i.c_2$ are the same field and α is different from β, the subset $S_{\alpha\vee\beta}$ contains those records that can have either α or β in the same field. Hence, the indicator polynomial is

$$P_{\{\alpha,\beta\}}(\mathbf{x}) = P_\alpha(\mathbf{x}) + P_\beta(\mathbf{x}). \quad (12)$$

Note that Equation (12) is different from Equation (11) due to the exclusive disjunction in the same field.

The construction in Equation (12) implies that an indicator polynomial can be constructed, providing a mechanism for counting a set of values that a record can contain.

In particular, to count the number of (integer) values within a range $[\alpha, \beta]$ in a countable field, $\mathcal{U}$ would construct the indicator polynomial as

$$P_{[\alpha,\beta]}(x) = \sum_{\chi=\alpha}^{\beta} P_\chi(x),$$

where $P_\chi(x)$ is the indicator polynomial of value χ within $[\alpha, \beta]$.

F. Optimization through arithmetization in $GF(2)$

EPiC's efficiency relies on the size of the ciphertexts, more precisely on the size of p. We refer to Section V for a detailed cost analysis. In this section, we discuss an optimization technique that significantly reduces the size of p while still supporting our counting functionality.

Suppose a data set contains n records with countable field $R_i.c \in \mathcal{D}$. A query to count occurrences of χ in $R_i.c$ requires an indicator polynomial, based on Equation (2), of degree $|\mathcal{D}| - 1$. Accordingly, the size of p needs to satisfy the requirement of Equation (8): $|p|$ increases linearly with the domain size $|\mathcal{D}|$.

Now, we treat the countable field $R_i.c \in \mathcal{D} > 2$ as a sequence of $m = \lceil \log_2 |\mathcal{D}| \rceil$ bits, where each bit is separately encrypted and stored in the cloud. In other words, we use m binary fields $\mathcal{D}_1 = \mathcal{D}_2 = \ldots = \mathcal{D}_m = \{0, 1\}$ to represent $\mathcal{D}$. Any counting query for a value $\chi \in \mathcal{D}$ can be equivalently transformed to a query on $\varphi = (x_1 = \chi_1 \wedge \ldots \wedge x_m = \chi_m)$, where x_i and χ_i represent the corresponding bits of x and χ.

A query on a conjunction of m fields requires the m-variate indicator polynomial over $GF(2)$, cf. Equation (9), as

$$P_\varphi(\mathbf{x}) = \textstyle\prod_{i=1}^{m} P_{\chi_i}(x_i), \quad P_{\chi_i}(x_i) = \begin{cases} 1 - x_i, & \text{if } x_i = 0 \\ x_i, & \text{if } x_i = 1 \end{cases}$$

The degree of $P_\varphi(\mathbf{x})$ is $\sum_{k=1}^{m}(|\mathcal{D}_k|-1) = \lceil \log_2 |\mathcal{D}| \rceil$, i.e., logarithmic in the domain size $|\mathcal{D}|$. Accordingly, the size of p, derived from Equation (10), increases only logarithmically to the domain size $|\mathcal{D}|$.

$$|p| \geq s_1 + \log_2 n + |q| + \lceil \log_2 |\mathcal{D}| \rceil \cdot (s_2 + |q|). \tag{13}$$

EPiC's evaluation in Section V will demonstrate that the above "transformation" of a single larger countable field with $\mathcal{D} > 2$ into equivalent multiple binary fields increases efficiency significantly.

Compared to the "Basic" protocol before, we will call the above optimization "GF(2) arithmetized" in our evaluation in Section V.

IV. Security Analysis

Lemma 1: Based on the Hidden Modular Group Order Assumption, EPiC's encryption scheme is IND-CPA.

Proof (Sketch): Our EPiC encryption is based on Trostle and Parrish's computationally Private Information Retrieval (cPIR) scheme [34]. Trostle and Parrish do not formally prove that their cPIR is IND-CPA secure, but they prove that it satisfies a formally defined security property (see their Definition 4.2). Their cPIR is shown to satisfy this security property under the Hidden Modular Group Order Assumption (HMGOA). They also provide evidence to support the claim of hardness of HMGOA. EPiC's IND-CPA security directly derives from the security property of their cPIR.

More precisely, the cPIR protocol is defined and proven to be secure, if, for any probabilistic polynomial time adversary $\mathcal{A}$, $\mathcal{A}$ cannot distinguish the least significant bits ("LSB") of a sequence of PIR values from uniform sampling by more than a negligible function of the key size. EPiC's encryption embeds the plaintext in the $|q|$ least significant bits.

In our notation, the cPIR security property can be stated as follows:

$$Pr[\mathcal{A} \text{ computes } LSB(\mathcal{C}) = m] \leq \frac{1}{q} + \epsilon(s) \tag{14}$$

for any plaintext m and ciphertext $\mathcal{C}$, where $LSB(\mathcal{C})$ are the $|q|$ least significant plaintext bits corresponding to ciphertext $\mathcal{C}$, and s is a sufficiently large security parameter. We re-formalize this notion of security by defining a simple game. Assume an adversary $\mathcal{A}$ can query an HMGOA oracle $\mathcal{O}_{\text{HMGOA}}$ by sending a bit string m. $\mathcal{O}_{\text{HMGOA}}$ randomly selects $b \in \{0,1\}$. If $b = 0$, then $\mathcal{O}_{\text{HMGOA}}$ randomly changes (at least one of) the low order bits of m and encrypts the result to $\mathcal{C}$ using Trostle and Parrish's scheme. If $b = 1$, the oracle just sends the encryption $\mathcal{C}$ of m back to $\mathcal{A}$. Finally, $\mathcal{A}$ has to decide whether $LSB(\mathcal{C}) = m$.

Now, assume that EPiC is not IND-CPA. Therefore, there exists a PPT adversary $\mathcal{A}'$ that on selecting two plaintexts (m_0, m_1) and given a ciphertext $\mathcal{C}_b = \text{Enc}(\mathcal{S}, m_b)$ encrypted by an EPiC encryption oracle $\mathcal{O}_{\text{EPiC}}$ with random $b \in \{0,1\}$ is able to guess whether $\mathcal{C}_b$ corresponds to m_0 or m_1 with a non-negligible probability advantage ϵ' over guessing. Such an adversary $\mathcal{A}'$ can now directly be used to break the security of Trostle and Parrish's cPIR, thus violating HMGOA, by constructing a new adversary $\mathcal{A}$ that employs $\mathcal{A}'$ as a subroutine.

$\mathcal{A}$ simulates $\mathcal{O}_{\text{EPiC}}$ to $\mathcal{A}'$ and receives two plaintexts (m_0, m_1). $\mathcal{A}$ randomly selects $m_b, b \in \{0,1\}$ and forwards it to $\mathcal{O}_{\text{HMGOA}}$, which sends back $\mathcal{C}$ to $\mathcal{A}$. This value $\mathcal{C}$ is again simply forwarded to $\mathcal{A}'$. If $\mathcal{A}'$ outputs a value $b' = b$, then $\mathcal{A}$ knows that $\mathcal{C}$ corresponds to (unmodified) m_b and outputs $LSB(\mathcal{C}) = m$. Otherwise, if $\mathcal{A}'$ aborts, then $\mathcal{A}$ aborts, too. $\mathcal{A}$ is successful in 50% of the cases, i.e., when $\mathcal{O}_{\text{HMGOA}}$ selects not to change a bit of m. $\mathcal{A}$ requires the same number of steps as $\mathcal{A}'$ and its advantage is $\epsilon = \frac{\epsilon'}{2}$, rendering our proof tight. Therewith, $\mathcal{A}$ is an adversary that efficiently breaks the scheme of Trostle and Parrish.

■

Lemma 2: Based on the Hidden Modular Group Order Assumption, EPiC is a privacy-preserving cloud counting scheme.

Proof (Sketch): This proof is based on the IND-CPA property of EPiC's encryption.

Our notion of privacy-preserving for cloud-based counting security is formalized in Definition 2. According to this definition we would need to prove two properties, the first corresponds to storage privacy, and the second corresponds to counting privacy. However, after proving storage privacy, we will show by reduction that storage privacy simply implies counting privacy, and we do not need to prove EPiC's counting privacy separately.

We consider the specific case of the detailed protocol described in Section III-D.

- **Storage Privacy:** The storage-privacy property is defined in the challenge phase of Game_1 as formalized in Algorithm 3.
  The adversary is supposed to guess the value of b when given $\{\mathcal{E}_b, Q_b, E_{\Sigma_b}\}$ where $\mathcal{E}_b := \text{Encrypt}(\mathcal{S}, \mathcal{R}_b)$, $Q_b := \text{PrepareQuery}(\mathcal{S}, \chi^b)$, and $E_{\Sigma_b} := \text{ProcessQuery}(Q_b, \mathcal{E}_b)$. Breaking the storage-privacy means that there exists a PPT adversary $\mathcal{A}$ who guesses bit b with a probability such that
  $$Pr(\text{Game}_1(\mathcal{A}) = 1) \leq \frac{1}{2} + \epsilon_1(s) \tag{15}$$
  is violated.
  However, note that Q_b is a sequence of independently IND-CPA encrypted coefficients. Also, $\mathcal{E}_b$ is the result of independent IND-CPA encryptions of $\mathcal{R}_b$. E_{Σ_b} can be computed by anyone who has access to $\mathcal{E}_b$ and therefore cannot give any additional information about b over $\mathcal{E}_b$. This is because of the specific somewhat homomorphic properties of EPiC encryption. Therefore, only Q_b and $\mathcal{E}_b$ can help $\mathcal{A}$ in guessing the value of b – but both are IND-CPA encrypted.
  Consequently, the storage-privacy property reduces to the IND-CPA property of EPiC's encryption, where $(\mathcal{R}_0, \chi^0)$ and $(\mathcal{R}_1, \chi^1)$ represent the selected plaintexts and $(\mathcal{E}_b, Q_b)$ the challenge ciphertext. The existence of

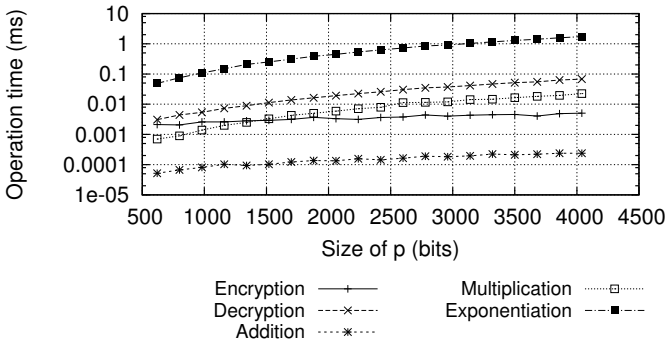

Fig. 1. Computation time of various operations on ciphertexts using our encryption scheme. The exponentiation time is measured for $\mathcal{C}^{15}$.

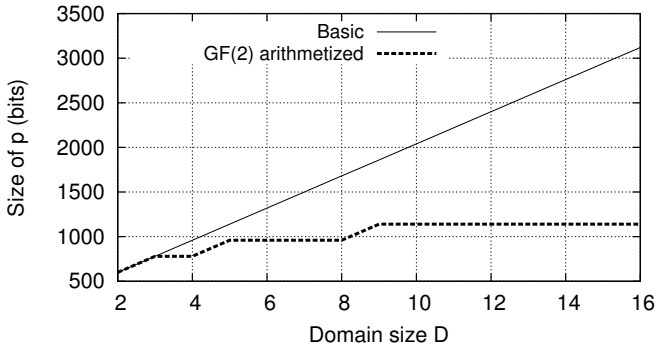

Fig. 2. Size of p depends on size of domain $\mathcal{D}$.

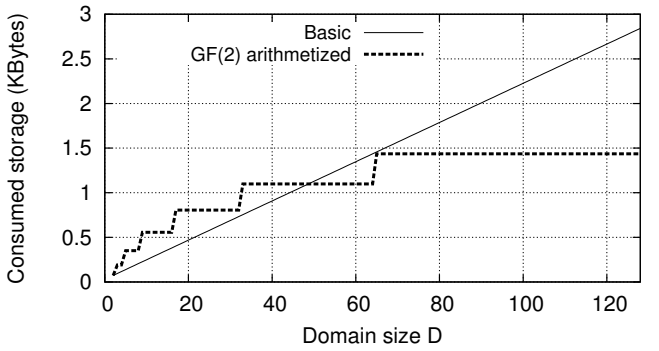

Fig. 3. Consumed storage for each record to store the countable field $\mathcal{D}$.

a PPT adversary who can guess bit b with a non-negligible probability would imply that EPiC is not IND-CPA secure.

- **"Storage Privacy $\Rightarrow$ Counting Privacy":** The counting privacy property is defined in the challenge phase of GAME_2 as formalized in Algorithm 3. We will now show by reduction that storage privacy implies counting privacy, i.e., any scheme that is storage private (such as EPiC) is also counting private.
  Assume there exists a PPT adversary $\mathcal{A}$ winning GAME_2 with a non-negligible probability advantage ϵ over guessing. Using this adversary $\mathcal{A}$ as a black box, we construct another PPT adversary $\mathcal{A}'$ that can win GAME_1 with a non-negligible probability advantage ϵ' over guessing. $\mathcal{A}'$ simulates the GAME_2 challenger for $\mathcal{A}$.
  As the learning phases for GAME_1 and GAME_2 are equivalent, $\mathcal{A}'$ simply forwards all requests $\{\mathcal{R}, \chi\}$ by $\mathcal{A}$ to the GAME_1 challenger and forwards answers $\{\mathcal{E}, Q, E_\Sigma, \sigma\}$ back to $\mathcal{A}$.
  For the challenge phase of $\mathcal{A}$ in GAME_2, $\mathcal{A}$ sends $\{\mathcal{R}, \chi^0, \chi^1\}, \chi^0 \neq \chi^1$, but $|\chi^0| = |\chi^1|$, to $\mathcal{A}'$. This is used by $\mathcal{A}'$ in the challenge phase of GAME_1 as follows: $\mathcal{A}'$ simply sends the distinct pair $\{(\mathcal{R}, \chi^0), (\mathcal{R}, \chi^1)\}$ to the GAME_1 challenger (i.e., $\mathcal{A}'$ sends the same $\mathcal{R}$ twice with different sets χ). This challenger sends back $\{\mathcal{E}, Q_b, E_{\Sigma_b}\}$ to $\mathcal{A}$, where $\mathcal{E}$ is the encryption of $\mathcal{R}$. $\mathcal{A}'$ forwards $\{\mathcal{E}, Q_b, E_{\Sigma_b}\}$ to $\mathcal{A}$ that outputs b, and $\mathcal{A}'$ also outputs b. If $\mathcal{A}'$ probability advantage in winning GAME_1 over guessing is $\epsilon' = \epsilon$, and $\mathcal{A}'$ needs the same number of steps as $\mathcal{A}$ rendering our reduction tight.

■

V. Evaluation

To show its real-world applicability, we have implemented and evaluated EPiC in the Hadoop MapReduce framework v1.0.3 [5]. The source code is available for download [4].

We have evaluated EPiC on Amazon's public MapReduce cloud [1]. Our EPiC implementation is written in Java, and all cryptographic operations are *unoptimized*, relying on Java's standard BigInteger data type. Still, exponentiation, e.g. $\mathcal{C}^j$, with $j = 15$ and $|\mathcal{C}| \approx 4000$ takes < 2ms on a 1.8GHz Intel Core i7 laptop, a single addition is not measurable with $< 1\mu$s. Figure 1 shows a benchmark of various operations on the ciphertexts using our encryption scheme. We would like to stress that the exponentiation, the most critical operation in our scenario, is two orders of magnitude faster than the one by Lauter et al. [22] – there, a single multiplication already consumes 40ms on a stronger CPU with 2.1GHz. In our evaluation, we use security parameters $s_1 = 400$ bits as suggested by Trostle and Parrish [34] for good security, and $s_2 = |r| = 160$ bits. We have implemented a data generator program to randomly generate patient records with m countable fields with size between 4 and 10 bits. As our evaluation targets comparing the performance of EPiC counting to non-private counting, we use encryption and decryption on the countable fields only. Other fields are not considered during the evaluation.

In this section, we evaluate the performance of EPiC by comparing our "Basic" and "GF(2) arithmetized" solutions with "non-privacy-perserving" solution.

For shorter presentation, we use "B" as index of the cost in Basic approach, and "G" in GF(2) arithmetized approach.

A. Size of prime p

While the prime q depends only on the number of records n, the prime p also depends on the domain size $|\mathcal{D}|$. For short notations, we set $u = s_1 + \log_2 n + |q|$, $v = s_2 + |q|$ and

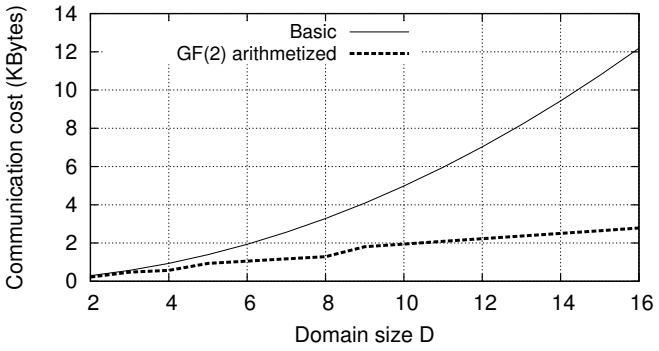

Fig. 4. Communication cost between the user and the cloud.

use them as fixed parameters (with respect to the domain size $|\mathcal{D}|$) when evaluating the cost. Derived from (8) and (13), the required size of p is

$$\begin{aligned} |p|_B &= u + (|\mathcal{D}| - 1) \cdot v \\ |p|_G &= u + \lceil \log_2 |\mathcal{D}| \rceil \cdot v. \end{aligned}$$

Figure 2 shows the logarithmic increase of $|p|$ with GF(2) arithmetized approach and linear increase with Basic approach.

B. Storage cost

The storage cost depends on the size of the data stored on the cloud. The larger the data is, the more the user has to pay for to the cloud, e.g. Amazon S3 [3]. In the following, we only focus on the size of the countable fields, which is determined by the size of prime p. Other fields are not considered.

Since the encrypted value in a countable field is an element in $GF(p)$, its size is at most $|p|$. In Basic approach, each record stores $|p|_B$ bits for the countable field. In GF(2) arithmetized approach, each record stores $\lceil \log_2 |\mathcal{D}| \rceil$ fields, each of size $|p|_G$. Therefore, the required storage is

$$\begin{aligned} S_B &= u + (|\mathcal{D}| - 1) \cdot v \\ S_G &= \lceil \log_2 |\mathcal{D}| \rceil \cdot (u + \lceil \log_2 |\mathcal{D}| \rceil \cdot v). \end{aligned}$$

Again, in Figure 3, we see a linear increase of storage in terms of the domain size $|\mathcal{D}|$ in Basic approach, while the GF(2) arithmetized approach consumption increase only logarithmically. This demonstrates a significant improvement on reducing the data size when "splitting" them into bit fields.

C. User computation and communication cost

1) Sending query: Due to oblivious counting, user $\mathcal{U}$ prepares and sends all coefficients corresponding to *all* monomials to the cloud. The number of generated and transferred coefficients is, therefore, the same for both Basic and GF(2) arithmetized approach, and always equal to the domain size $|\mathcal{D}|$.

Computation: Since the computation of coefficients is done by $\mathcal{U}$ on plaintexts, and the encryption is fast (see Figure 1), the computation cost at the user side in both approaches is negligible compared to the cloud computation cost.

Communication: The transferred size of the encrypted coefficients, however, is $|\mathcal{D}|$ times the size of p. Consequently, the query size is

$$\begin{aligned} Q_B &= |\mathcal{D}| \cdot (u + (|\mathcal{D}| - 1) \cdot v) \\ Q_G &= |\mathcal{D}| \cdot (u + \lceil \log_2 |\mathcal{D}| \rceil \cdot v). \end{aligned}$$

As an example, for a data set containing $n = 10^6$ records with a countable field of domain size $|\mathcal{D}| = 1024$, the query size is $Q_B = 22.5$MBytes, and $Q_G = 280$KBytes.

2) Receiving answer:

Computation: When receiving the encrypted sum E_Σ, $\mathcal{U}$ only needs to decrypt only one ciphertext to obtain the count value. Since the decryption time is fast (see Figure 1) compared to the cloud computation time, the user's processing time is negligible.

Communication: The size of the received ciphertext increases during the cloud computation due to series of additions and multiplications.

In the Basic approach (Algorithm 4), for each record R_i, the Mappers compute the monomials by the exponentiation $(E_{c_i})^j$, $0 \le j \le |\mathcal{D}| - 1$. The ciphertext size will increase to roughly $(|\mathcal{D}| - 1) \cdot |p|_B$ bits. Since the additions $\sum_{i=1}^{n} (E_{c_i})^j$ across n records only add $\log_2 n$ bits, which is negligible to $|p|_B$, the ciphertext returned to the Reducer is $(|\mathcal{D}| - 1) \cdot |p|_B$ bits long. At the Reducer, the evaluation is performed by one multiplication between the sums of monomials with the corresponding coefficients, therefore, adding $|p|_B$ bits to the ciphertext, which results in the final ciphertext size of roughly $|\mathcal{D}| \cdot |p|_B$ bits. This is approximately equal to the query size Q_B.

In binary fields counting approach (Algorithm 5), for each record R_i, the Mappers compute the monomials by $\prod_{k=1}^{m} E_{c_{i,k}}^{j_k}$, in which $j_k \in \{0, 1\}$, $m = \lceil \log_2 |\mathcal{D}| \rceil$. Therefore, the number of multiplications to compute monomials is at most $m - 1$, and so, the ciphertext size may increase to at most $(m - 1) \cdot |p|_G$ bits. Again, due to insignificant size increase by the additions and due to one more multiplication at the Reducer, the resulted ciphertext size will increase to $m \cdot |p|_G = \lceil \log_2 |\mathcal{D}| \rceil \cdot |p|_G$ bits.

In conclusion, the received answer size is

$$\begin{aligned} A_B &= |\mathcal{D}| \cdot (u + (|\mathcal{D}| - 1) \cdot v) \\ A_G &= \lceil \log_2 |\mathcal{D}| \rceil \cdot (u + \lceil \log_2 |\mathcal{D}| \rceil \cdot v). \end{aligned}$$

As an example, for a data set containing $n = 10^6$ records with a countable field of domain size $|\mathcal{D}| = 1024$, the answer size is $A_B = 22.5$Mbytes, and $A_G = 2.7$KBytes.

Consequently, the total communication cost $(C = Q + A)$ with multiple binary fields is much less for the same amount of carried information (Figure 4).

$$\begin{aligned} C_B &= 2 \cdot |\mathcal{D}| \cdot (u + (|\mathcal{D}| - 1) \cdot v) \\ C_G &= (|\mathcal{D}| + \lceil \log_2 |\mathcal{D}| \rceil) \cdot (u + \lceil \log_2 |\mathcal{D}| \rceil \cdot v). \end{aligned}$$

D. Cloud computation

We evaluate the cloud computation cost for large-scale data sets on Amazon's public cloud. As Amazon imposes an (initial) limit of 20 instances per job, we restrict ourselves to 20 Standard Large On-Demand instances [2]. Each instance comprises 4 processors with 2.27GHz Intel Xeon CPU, 4MB of cache, and total 7.5GB of memory.

1) Variable data set size: Coming back to our example application scenario with patient records, in a first experiment, we fix the size of each record to 1MB. In the real world, this would reflect to a patient record that might, besides doctoral notes and prescription, also comprise, e.g., an X-ray picture. The data set size (x-axis) is varied from 100GB to 1TB. In this experiment, we query a countable field of size $|\mathcal{D}| = 16$. Figure 5 shows the average counting time for a MapReduce job on the whole data set of different sizes. The y-axis shows the total time for MapReduce to evaluate the user's query. This is the time that a user has to pay for to, e.g., Amazon [2]. To put our results into perspective, we not only show the time for both Basic and GF(2) arithmetized approaches, but as well the time a "non-privacy-preserving" counting would take, i.e., the countable field is not encrypted and directly counted.

Moreover, we also show the overhead ratio between EPiC's two approaches and non-private counting. The additional overhead introduced by EPiC over non-private counting is less than 20%. As Amazon's pricing scales directly proportional with the CPU time. This additional overhead would also be the additional amount of money user $\mathcal{U}$ would have to pay Amazon. We conjecture that only 20% overhead/additional cost over non-privacy-preserving counting is acceptable in many real-world situations, rendering EPiC practical.

2) Variable record size: To also evaluate the effect of the size of the records on the general performance, we run the system with a fixed data set size of 50GB. The record size is changed from 100KB to 1MB. Figure 6 shows that, while IO time remains unchanged, a higher number of records increases counting time in EPiC. However, the overhead of EPiC is still under 20% even for small record sizes such as 100KB compared to non-private counting. That is, EPiC is efficient even for small patient records.

3) Effect of multiple fields: To study the efficiency of transforming a countable field $\mathcal{D}$ into multiple subfields of different size, we conduct an experiment on a data set size of 100GB. The data set contains a countable field of domain size $|\mathcal{D}| = 1024$ (10 bits). We compare three cases: (a) transform $\mathcal{D}$ into 10 single bit fields; (b) transform $\mathcal{D}$ into 5 fields of 2 bits; (c) transform $\mathcal{D}$ into 3 fields of 3 bits, 3 bits, and 4 bits. In Figure 7, we can see that the GF(2) arithmetized approach yields the best performance.

4) Multiple counting: Finally, to evaluate the effects of different query types on the performance of EPiC, we run EPiC with a fixed data set of 100GB. The total domain size is $|\mathcal{D}| = 1024$. We make 3 different queries: (a) query for a specific value; (b) query for values in the range $[0, \frac{|\mathcal{D}|}{2}]$, i.e., looking for the MSB of the field equal to 0; (c) query for all even values in the field, i.e., looking for the LSB of the field equal to 0. As we can see in Figure 8, there is no significant difference in counting time between different queries.

VI. Related Work

Protecting the privacy of outsourced data and delegated operations in a cloud computing environment is the perfect setting for fully homomorphic encryption. While there is certainly a lot of ongoing research in fully homomorphic encryption (see Lipmaa [23] for an overview), current implementations indicate high storage and computational overhead [14, 26]. This renders fully homomorphic encryption impractical for cloud computing.

Similar to EPiC, Lauter et al. [22] observe that, depending on the application, weaker "somewhat" homomorphic encryption might be sufficient. Lauter et al. [22]'s scheme is based on a protocol for lattice-based cryptography by Brakerski and Vaikuntanathan [10]. However, for the application scenario considered in this paper, EPiC's somewhat homomorphic encryption scheme allows for much faster exponentiation.

Our work bears some similarity with the work of Kamara and Raykova [21]. That paper develops fully hiding algorithms to evaluate uni/multi-variate polynomials at a given point. In particular, they develop mechanisms that preserve the secrecy of the evaluation point though randomized reducibility techniques, and even the secrecy of the polynomial using somewhat homomorphic schemes that allow for one multiplication and an arbitrary subsequent number of additions (e.g., 2DNF-HE [9]). However, their techniques do not solve the problem where the evaluation point is stored on the cloud indistinguishably encrypted. EPiC exploits a property of the Trostle and Parrish [34] scheme to allow for exponentiation followed by an arbitrary number of additions.

Other research has addressed similar problems of performing operations on outsourced data, such as privacy-preserving searching on encrypted data [8, 25, 31, 35]. While searching and counting might be closely related, it is far from straightforward to adopt these schemes to perform efficient counting in a highly parallel cloud computing, e.g., MapReduce environment. Also notice that, e.g., Boneh et al. [8] rely on the computation of very expensive bilinear pairings for each element of a data set, rendering this approach impractical in a cloud setting.

Much research has been done to compute statistics in a privacy-preserving manner using *differential privacy*, see the seminal paper by Dwork [12]. Contrary to the threat model considered in this paper, the adversary is not the infrastructure (a database in their case), but a curious adversary trying to learn information about individual entries in the database. The idea of differential privacy is to add noise to aggregated query results. In this context, also the application of differential privacy to MapReduce clouds has been investigated [29]. EPiC, however, addresses the opposite problem where a user does not trust the cloud infrastructure.

VII. Conclusion

In this paper, we presented EPiC to address a fundamental problem of statistics computation on outsourced data: privacy-

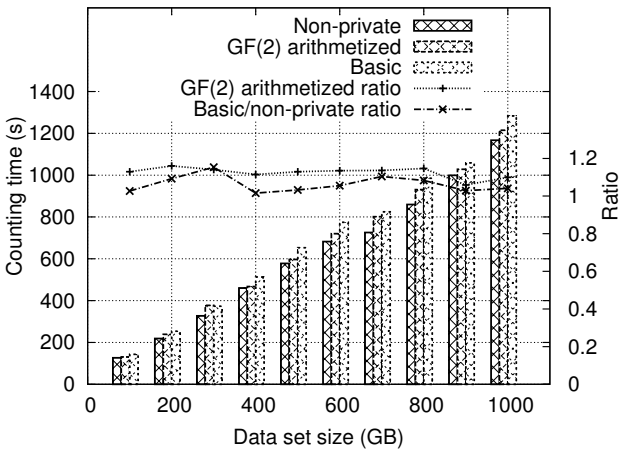

Fig. 5. Counting time versus data set size. $|\mathcal{D}| = 16$.

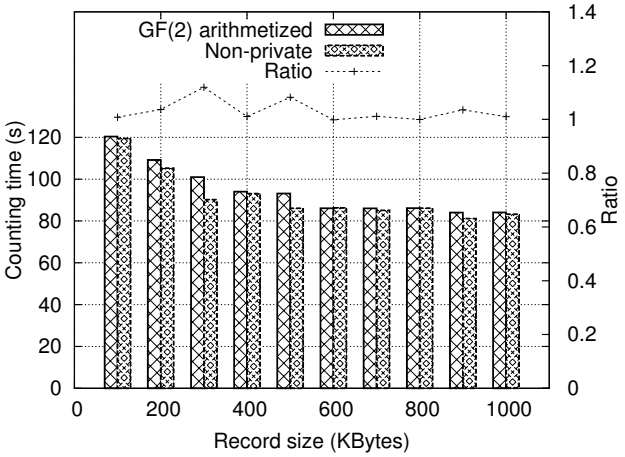

Fig. 6. 50GB data sets, varying record size. $|\mathcal{D}| = 16$.

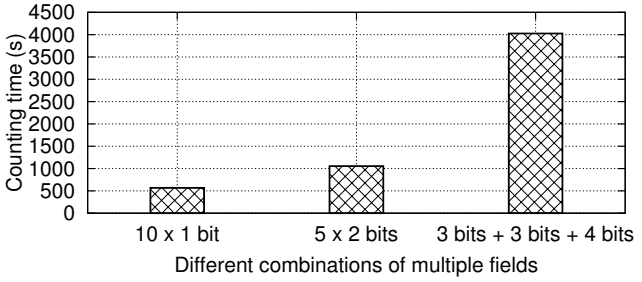

Fig. 7. Effect of different field combinations.

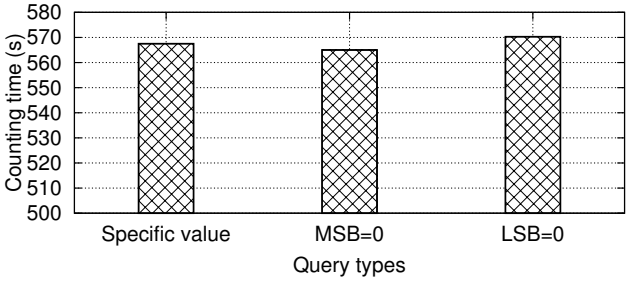

Fig. 8. Different query types on the same data set.

preserving pattern counting. EPiC's main idea is to count occurrences of patterns in outsourced data through a privacy-preserving summation of the pattern's indicator-polynomial evaluations over the encrypted dataset records. Using a "somewhat homomorphic" encryption mechanism, the cloud neither learns any information about outsourced data nor about the queries performed. Our implementation and evaluation results for MapReduce running on Amazon's cloud with up to 1 TByte of data show only modest overhead compared to non-privacy-preserving counting. Contrary to related work, this makes EPiC practical in a real-world cloud computing setting today.

REFERENCES

[1] Amazon. Elastic MapReduce, 2012. http://aws.amazon.com/elasticmapreduce/.

[2] Amazon. Amazon Elastic MapReduce Price, 2012. http://aws.amazon.com/elasticmapreduce/#pricing.

[3] Amazon. Amazon Simple Storage Service (Amazon S3) Price, 2012. http://aws.amazon.com/s3/#pricing.

[4] Anonymized. EPiC Source Code, 2012.

[5] Apache. Hadoop, 2010. http://hadoop.apache.org/.

[6] L. Babai and L. Fortnow. Arithmetization: A New Method In Structural Complexity Theory. *Computational Complexity*, pages 41–66, 1991. ISSN 1016-3328.

[7] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Proceedings of Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, USA, 1997. ISBN.

[8] D. Boneh, G. DiCrescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proceedings of Eurocrypt*, pages 506–522, Barcelona, Spain, 2004.

[9] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of International Conference on Theory of Cryptography*, pages 325–341, Cambridge, USA, 2005. ISBN 3-540-24573-1.

[10] Z. Brakerski and V. Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Proceedings of Annual Cryptology Conference*, pages 505–524, Santa Barbara, USA, 2011. ISBN 978-3-642-22791-2.

[11] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, USA, 2004.

[12] C. Dwork. Differential Privacy. In *Proceedings of Colloquium Automata, Languages and Programming*, pages 1–12, Venice, Italy, 2006. ISBN 3-540-35907-9.

[13] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of ACM Symposium on Theory of*

Computing, pages 169–178, Bethesda, USA, 2009. ISBN 978-1-60558-506-2.

[14] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Proceedings of International Conference on Theory and Applications of Cryptographic Techniques*, pages 129–148, Tallinn, Estonia, 2011. ISBN 78-3-642-20464-7.

[15] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. ISSN 0022-0000.

[16] Google. A new approach to China, 2010. http://googleblog.blogspot.com/2010/01/new-approach-to-china.html.

[17] Google. Google App Engine API for running MapReduce jobs, 2011. http://code.google.com/p/appengine-mapreduce/.

[18] Hadoop. Powered by Hadoop, list of applications using Hadoop MapReduce, 2011. http://wiki.apache.org/hadoop/PoweredBy.

[19] C. Hall, I. Goldberg, and B. Schneier. Reaction attacks against several public-key cryptosystems. In *Proceedings of International Conference on Information and Communication Security*, pages 2–12, Sydney, Australia, 1999. ISBN 3-540-66682-6.

[20] IBM. InfoSphere BigInsights, 2011. http://www-01.ibm.com/software/data/infosphere/biginsights/.

[21] S. Kamara and M. Raykova. Parallel Homomorphic Encryption. Technical Report, ePrint Report 2011/596, 2011. http://eprint.iacr.org/2011/596.

[22] K. Lauter, N. Naehrig, and V. Vaikuntanathan. Can Homomorphic Encryption be Practical? In *Proceedings of ACM Workshop on Cloud Computing Security*, Chicago, USA, 2011. ISBN 978-1-4503-1004-8.

[23] H. Lipmaa. Fully-Homomorphic Encryption, 2012. http://www.cs.ut.ee/~lipmaa/crypto/link/public/fhe.php.

[24] Microsoft. Daytona, 2011. http://research.microsoft.com/en-us/projects/daytona/.

[25] V. Pappas, M. Raykova, B. Vo, S.M. Bellovin, and T. Malkin. Private Search in the Real World. In *Proceedings of Computer Security Applications Conference*, pages 83–92, Orlando, USA, 2011. ISBN 978-1-4503-0672-0.

[26] H. Perl, M Brenner, and M. Smith. An Implementation of the Fully Homomorphic Smart-Vercauteren Crypto-System. In *Proceedings of Conference on Computer and Communications Security*, pages 837–840, Chicago, USA, 2011. ISBN 978-1-4503-0948-6.

[27] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 199–212, Chicago, USA, 2009. ISBN 978-1-60558-894-0.

[28] F. Rocha and M. Correia. Lucy in the Sky without Diamonds: Stealing Confidential Data in the Cloud. In *Proceedings of International Workshop on Dependability of Clouds, Data Centers and Virtual Computing y Environments*, Hong Kong, China, 2011. http://www.gsd.inesc-id.pt/~mpc/pubs/diamonds-final.pdf.

[29] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and Privacy for MapReduce. In *Proceedings of Symposium on Networked Systems Design and Implementation*, pages 297–312, San Jose, USA, 2010. ISBN 978-931971-73-7.

[30] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4): 869–877, 1992. ISSN 0004-5411.

[31] D.X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of Symposium on Security and Privacy*, pages 44–55, Berkeley, USA, 2000.

[32] Techcrunch. Google Confirms That It Fired Engineer For Breaking Internal Privacy Policies, 2010. http://techcrunch.com/2010/09/14/google-engineer-spying-fired/.

[33] The Telegraph. Patient records go online in data cloud, 2011. http://www.telegraph.co.uk/health/healthnews/8600080/Patient-records-go-online-in-data-cloud.html.

[34] J. Trostle and A. Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In *Proceedings of Conference on Information Security*, pages 114–128, Boca Raton, USA, 2010.

[35] C. Wang, K. Ren, S. Yu, and K.M.R. Urs. Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data. In *Proceedings of International Conference on Computer Communications*, pages 451–459, Orlando, USA, 2012. ISBN 978-1-4673-0773-4.

[36] Z. Whittaker. Microsoft admits Patriot Act can access EU-based cloud data. Zdnet, 2011. http://www.zdnet.com/.