EPiC: Efficient Privacy-Preserving Counting for MapReduce

Erik-Oliver Blass   Guevara Noubir   Triet Vo-Huu
College of Computer and Information Science
Northeastern University, Boston, MA 02115
{blass|noubir|vohuudtr}@ccs.neu.edu

***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. EPiC's main idea is, first, to reduce the problem of counting to polynomial evaluation. Second, to efficiently evaluate polynomials in a privacy-preserving manner, we extend previous work on the Hidden Modular Group Order assumption and design a new *somewhat homomorphic* encryption scheme. This scheme is highly efficient in our particular counting scenario with a relatively small number of possible patterns. 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 [9]. MapReduce is offered by major public cloud providers today, such as Amazon [1], Google [15], IBM [18] or Microsoft [22]. 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 [16].

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 [14, 25, 26, 29, 33]. 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 [11] or see Lipmaa [21] 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 [12, 24], and a deployment in a real-world cloud would outweigh any cost advantage offered by the cloud. Moreover, 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. As of today, it is unclear (and far from being straightforward) how fully homomorphic encryption could be tailored to adopt to this paradigm. In conclusion, fully homomorphic encryption is impractical for privacy-preserving cloud computing.

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 comprises 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. 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 counting to an evaluation of a special polynomial. Inspired by Lauter et al. [20], EPiC evaluates this polynomial efficiently by using a novel, efficient *somewhat homomorphic* encryption mechanism that is based on the Hidden Modular Group Order assumption [31] – fully homomorphic encryption is not required. The new encryption mechanism is efficient only in the particular context that we target in this paper, the evaluation of polynomials 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 counting in MapReduce clouds. EPiC reduce the problem of counting occurrences of patterns to the evaluation of special polynomials that the cloud user can specify.
- A new "somewhat homomorphic" IND-CPA encryption scheme that addresses the secure evaluation of polynomials 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 [3].

II. Problem Statement

Overview: We will use an example application to motivate our work. Along the lines of recent reports [30], 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) field $R.c$ 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 rest of this paper, we will assume the "pattern" a user $\mathcal{U}$ might look for to be "countable". Without loss of generality, field $R.c \in \mathbb{N}$.

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 a countable field $R_i.c \in \mathbb{N}$. A privacy-preserving counting scheme comprises the following probabilistic polynomial time algorithms:

1) $\textsc{KeyGen}(s)$: using a security parameter s, $\textsc{KeyGen}$ outputs a secret key $\mathcal{S}$
2) $\textsc{Encrypt}(\mathcal{S}, \mathcal{R})$: 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) $\textsc{Upload}(\mathcal{E})$: uploads the sequence of encryptions $\mathcal{E}$ to the cloud.
4) $\textsc{PrepareQuery}(\mathcal{S}, \chi)$: this algorithms generates a query Q out of secret $\mathcal{S}$ and the value $\chi \in \mathbb{N}$ to be counted.
5) $\textsc{ProcessQuery}(Q, \mathcal{E})$: performs the actual counting. Uses a query Q, the sequence of ciphertexts $\mathcal{E}$, and outputs a result E_Σ.
6) $\textsc{Decode}(\mathcal{S}, E_\Sigma)$: takes secret $\mathcal{S}$ and query result E_Σ to output a final sum σ, such that $\sigma = \sum_{i=1}^{n} f_\chi(R_i.c)$, if $E_\Sigma = \textsc{ProcessQuery}(Q, \mathcal{E})$ with $Q = \textsc{PrepareQuery}(\mathcal{S}, \chi)$, $\mathcal{E} = \textsc{Encrypt}(\mathcal{S}, \mathcal{R})$, and

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

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 $\textsc{KeyGen}$, $\textsc{Encryt}$, $\textsc{Upload}$, $\textsc{PrepareQuery}$, and $\textsc{Decode}$ should be lightweight compared to $\textsc{ProcessQuery}$.

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 [13], 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, we stick to the separated setup for ease of understanding.

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:   $\mathcal{R} := \{R_1, \ldots, R_n\}$;
    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.

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 $|\mathcal{R}_0| = |\mathcal{R}_1|$ 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 2: Challenge Phase $\text{GAME}_1(\mathcal{A})$

```
    $\mathcal{A} \rightarrow$ Challenger:   $(\mathcal{R}_0, \chi_0)$, $(\mathcal{R}_1, \chi_1)$, $|\mathcal{R}_0| = |\mathcal{R}_1|$;
    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;
```

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

```
    $\mathcal{A} \rightarrow$ Challenger:   $\mathcal{R}$, $(\chi_0, \chi_1)$;
    Challenger:                $b \leftarrow \{0, 1\}$;
                               $\mathcal{E} := \text{ENCRYPT}(\mathcal{S}, \mathcal{R})$;
                               $Q_b := \text{PREPAREQUERY}(\mathcal{S}, \chi_b)$;
                               $E_{\Sigma_b} := \text{PROCESSQUERY}(\mathcal{S}, \chi_b)$;
    Challenger$\rightarrow \mathcal{A}$ :  $\{Q_b, E_{\Sigma_b}\}$;
    $\mathcal{A}$:                     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 queries (χ_0, χ_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 $E_{\Sigma_b} := \text{PROCESSQUERY}(\mathcal{S}, \chi_b)$. The challenger sends $\{Q_b, E_{\Sigma_b}\}$ back to $\mathcal{A}$. Note that $\mathcal{A}$ does not receive $\mathcal{E}$. $\mathcal{A}$ 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*

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

for all probabilistic polynomial time adversaries $\mathcal{A}$ with running time T. Functions $\epsilon_1(s)$ and $\epsilon_2(s)$ are negligible, i.e., $\epsilon_1(s) < \frac{1}{s^{n_1}}$ and $\epsilon_2(s) < \frac{1}{s^{n_2}}$ 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 $\textsc{Game}_2$, $\mathcal{A}$'s goal is, by using the same means as in $\textsc{Game}_1$, 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 $\textsc{Game}_1$. Otherwise, just by looking at the size of the encrypted records, $\mathcal{A}$ could win $\textsc{Game}_1$. 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" [17]. 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 [9].

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

Overview: EPiC's main rationale is to perform the counting in the cloud by evaluating a polynomial. Cloud user $\mathcal{U}$ sends a polynomial $P_\chi(x)$ to the cloud that is specific to the value χ he is interested in. The cloud evaluates $P_\chi(x)$ on each stored record's countable value $R_i.c$. The outcomes of all individual polynomial evaluations is a (large) set of monomials. The cloud now simply adds monomials with the same exponent and sends the sums back to $\mathcal{U}$. Based on the received sums of monomials, $\mathcal{U}$ can learn the number of occurrences of χ in the investigated set of records.

There are three challenges with this approach.

First, the cloud has to evaluate polynomial $P_\chi(x)$ on encrypted data. More precisely, to avoid the cloud to learn 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, 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 every 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. [20]. 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 to compute monomials, the overhead remains modest – see Section IV. 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. We prove security of EPiC based on the Hidden Modular Group Order assumption [31].

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 [5].

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 monomials with the same exponent can finally be done by Reducers within the reduce phase.

A. Polynomial Counting

Besides other data, each patient record R_i in EPiC contains a countable field $R_i.c$. If $\mathcal{U}$ wishes to count occurrences of χ, the idea is that $\mathcal{U}$ prepares a polynomial

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

Therewith, the cloud can scan through the set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of all patient records and compute $\sum_{i=1}^{n} P_\chi(R_i.c)$.

If we assume the domain for χ and $R_i.c$ to be $\mathcal{D} = \{0, 1, 2, \ldots, |\mathcal{D}| - 1\} \subseteq \mathbb{N}$, then one way to generate a polynomial $P_\chi(x)$ is to compute

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

where x_j are all possible elements of $\mathcal{D}$. The polynomial $P_\chi(x)$ is of degree $|\mathcal{D}| - 1$ and uniquely defined by its coefficients a_j.

However, the countable fields are IND-CPA encrypted to $\textsc{Enc}(R_i.c)$ in EPiC using the somewhat homomorphic encryption. One way to evaluate $P_\chi(x)$ in an oblivious manner would be to send (IND-CPA) encrypted coefficients $\textsc{Enc}(a_j)$ to the cloud. The cloud would now compute the "encrypted" sum

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

Yet, for improved performance, EPiC chooses a slightly different approach: for each record $\textsc{Enc}(R_i)$ with encrypted countable field $\textsc{Enc}(R_i.c)$ the cloud simply computes the set of all $|\mathcal{D}|$ monomials $\{1, \textsc{Enc}(R_i.c), \textsc{Enc}(R_i.c)^2, \ldots, \textsc{Enc}(R_i.c)^{|\mathcal{D}|-1}\}$ and sends back to $\mathcal{U}$ the $|D|$ sums

$$\{n, \textstyle\sum_{i=1}^{n} \textsc{Enc}(R_i.c), \sum_{i=1}^{n} (\textsc{Enc}(R_i.c))^2, \\ \ldots \\ \textstyle\sum_{i=1}^{n} (\textsc{Enc}(R_i.c))^{|\mathcal{D}|-1}\}.$$

User $\mathcal{U}$ can now decrypt $E_{\Sigma_j} = \sum_{i=1}^{n} (\textsc{Enc}(R_i.c))^j$ to get plaintext sums of monomials $\Sigma_j := \sum_{i=1}^{n} (R_i.c)^j, 0 \leq j \leq |\mathcal{D}| - 1$. Finally, $\mathcal{U}$ computes

$$\sigma := \sum_{j=0}^{|\mathcal{D}|-1} a_j \cdot \Sigma_j = P_\chi(x).$$

Note that, if $|\mathcal{D}|$ is reasonably small, the sums E_{Σ_j} can be evaluated efficiently by the cloud, see Section IV.

B. 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 main idea of our encryption scheme is borrowed from the Hidden Modular Group Order assumption scheme by Trostle and Parrish [31]. To encrypt a plaintext $\mathcal{P}$, a random number r is selected and, together with some system parameter η, added to $\mathcal{P}$, i.e., $r \cdot 2^\eta + \mathcal{P}$. Finally, a random parameter $b \in \mathbb{Z}_p$ for some large prime p is multiplied, resulting in ciphertext $\mathcal{C} := b \cdot (r \cdot 2^\eta + \mathcal{P}) \bmod p$. This setup has the interesting property that the random $r \cdot 2^\eta$ can be "canceled out" during decryption by computing $\bmod 2^\eta$. However before, b is removed by multiplying with $b^{-1} \bmod p$. Informally speaking, the Hidden Modular Group Order assumption by Trostle and Parrish [31] 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 III-D.

We choose this special somewhat homomorphic scheme simply due to its efficiency for the must crucial operation: the computation of monomials in the cloud. As the cloud has to scan through a large amount of records, it needs to compute a huge number of monomials. The way we perform the exponentiation for monomials is around two orders of magnitude faster than the one by Lauter et al. [20].

Again, we stress that our scheme is not a "general" homomorphic encryption scheme, because ciphertexts increase (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 exponentiations performed. This information, too, is only available in our special scenario.

Description

EPiC's somewhat homomorphic encryption is defined by the following set of algorithms.

- $\textsc{KeyGen}(s_1, s_2, n, |\mathcal{D}|)$: 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 $|\mathcal{D}| \in \mathbb{N}$ is the size of the domain of the countable values fields. $\textsc{KeyGen}$ computes
  1) value $\eta := \lceil \log_2 n + (|\mathcal{D}| - 1) \cdot \log_2(|\mathcal{D}| - 1) \rceil$.
  2) a random large prime p, where
  $$|p| = s_1 + \lceil \log_2 n \rceil + (s_2 + \eta)(|\mathcal{D}| - 1).$$
  3) a random $b \in \mathbb{Z}_p$.

  We explain the selection of η and p further below.
  The secret key, the output of $\textsc{KeyGen}$, is defined as $\mathcal{S} := \{p, b\}$.
- $\textsc{Enc}(\mathcal{S}, \mathcal{P})$: Select random number $r, |r| = s_2$. The plaintext $\mathcal{P}$ is encrypted to ciphertext
  $$\mathcal{C} := b \cdot (r \cdot 2^\eta + \mathcal{P}) \bmod p.$$

- $\textsc{Add}(\mathcal{C}_1, \mathcal{C}_2)$: Compute
$$\mathcal{C}' := \mathcal{C}_1 + \mathcal{C}_2.$$
This addition takes place in the integers, there is no modulo reduction. Consequently, the length of the ciphertext increases to:
$$|\mathcal{C}'| = \begin{cases} |\mathcal{C}_1| + 1, & \text{if } |\mathcal{C}_1| = |\mathcal{C}_2| \\ \max(|\mathcal{C}_1|, |\mathcal{C}_2|), & \text{otherwise} \end{cases}$$
- $\textsc{Exponentiate}(\mathcal{C}, j)$: With $j \in \mathbb{N}$, compute
$$\mathcal{C}' := \mathcal{C}^j.$$
Again, note that this is integer exponentiation, there is no modulo computation, and the ciphertext length increases to $|\mathcal{C}'| := j \cdot (|\mathcal{C}| - 1) + 1$.
- $\textsc{Dec}(\mathcal{S}, \mathcal{C}^j, j)$: To decrypt, compute $b^{-j} \cdot \mathcal{C}^j \bmod p \bmod 2^\eta$. Notice the soundness of our protocol:
$$\begin{aligned} b^{-j} \cdot \mathcal{C}^j \bmod p \bmod 2^\eta &= b^{-j} \cdot [b \cdot (r \cdot 2^\eta + \mathcal{P})]^j \\ &\quad \bmod p \bmod 2^\eta \\ &= (r \cdot 2^\eta + \mathcal{P})^j \bmod 2^\eta \\ &= \mathcal{P}^j. \end{aligned}$$
We assume that the decryption mechanism "knows" the exponent j of the ciphertext. This is a valid assumption in our special scenario where the cloud returns the (ordered) sequence of encrypted monomials. So, $\mathcal{U}$ knows the exponent $0 \le j \le |\mathcal{D}| - 1$ for each ciphertext.

Homomorphic Properties

Our scheme is additively homomorphic:
$\textsc{Dec}(\mathcal{C}_1^j + \mathcal{C}_2^j, j)$
$$\begin{aligned} &= b^{-j} \cdot [b^j \cdot (r_1 \cdot 2^\eta + \mathcal{P}_1)^j + b^j \cdot (r_2 \cdot 2^\eta + \mathcal{P}_2)^j] \\ &\quad \bmod p \bmod 2^\eta \\ &= [(r_1 \cdot 2^\eta + \mathcal{P}_1)^j + (r_2 \cdot 2^\eta + \mathcal{P}_2)^j] \bmod p \bmod 2^\eta \\ &= b^{-j} \cdot (b^j \cdot (r_1 \cdot 2^\eta + \mathcal{P}_1)^j) \bmod p \bmod 2^\eta \\ &\quad + b^{-j} \cdot (b^j \cdot (r_2 \cdot 2^\eta + \mathcal{P}_2)^j) \bmod p \bmod 2^\eta \\ &= \textsc{Dec}(\mathcal{C}_1^j, j) + \textsc{Dec}(\mathcal{C}_2^j, j) \\ &= \mathcal{P}_1^j + \mathcal{P}_2^j. \end{aligned}$$
Note that the third equality above holds for all $r_1, r_2, \mathcal{P}_1, \mathcal{P}_2$, only if
$$(r_1 \cdot 2^\eta + \mathcal{P}_1)^j + (r_2 \cdot 2^\eta + \mathcal{P}_2)^j < p$$
or
$$(b^{-1} \cdot \textsc{Enc}(\mathcal{S}, \mathcal{P}_1))^j + (b^{-1} \cdot \textsc{Enc}(\mathcal{S}, \mathcal{P}_2))^j < p. \quad (1)$$
Moreover, our scheme is multiplicatively homomorphic:

$\textsc{Dec}(\mathcal{C}_1^i \cdot \mathcal{C}_2^j, i + j)$
$$\begin{aligned} &= b^{-(i+j)} \cdot [b^i \cdot (r_1 \cdot 2^\eta + \mathcal{P}_1)^i \cdot b^j \cdot (r_2 \cdot 2^\eta + \mathcal{P}_2)^j] \\ &\quad \bmod p \bmod 2^\eta \\ &= [(r_1 \cdot 2^\eta + \mathcal{P}_1)^i \cdot (r_2 \cdot 2^\eta + \mathcal{P}_2)^j] \bmod p \bmod 2^\eta \\ &= b^{-i} \cdot [b^i \cdot (r_1 \cdot 2^\eta + \mathcal{P}_1)^i \bmod p \bmod 2^\eta \\ &\quad \cdot b^{-j} \cdot [b^j \cdot (r_2 \cdot 2^\eta + \mathcal{P}_2)^j \bmod p \bmod 2^\eta e \\ &= \textsc{Dec}(\mathcal{C}_1^i, i) \cdot \textsc{Dec}(\mathcal{C}_2^j, j) \\ &= P_1^i \cdot P_2^j. \end{aligned}$$
Again, the third equality above holds for all $r_1, r_2, \mathcal{P}_1, \mathcal{P}_2$, only if
$$(r_1 \cdot 2^\eta + \mathcal{P}_1)^i \cdot (r_2 \cdot 2^\eta + \mathcal{P}_2)^j < p$$
or
$$(b^{-1} \cdot \textsc{Enc}(\mathcal{S}, \mathcal{P}_1))^i \cdot (b^{-1} \cdot \textsc{Enc}(\mathcal{S}, \mathcal{P}_2))^j < p \quad (2)$$
Conditions in (1) and (2) have impact on the selection of p.

How to select p

The above conditions for the additive and multiplicative homomorphic properties imply inequality
$$\sum_{i=1}^{n} (b^{-1} \cdot \textsc{Enc}(\mathcal{S}, R_i.c))^j < p$$
which must be valid for all values of $R_i.c, j$, and r_i as chosen during encryption. This yields
$$p > n \cdot ((2^{s_2} - 1) \cdot 2^\eta + |\mathcal{D}| - 1)^{|\mathcal{D}|-1}. \quad (3)$$
Since $|\mathcal{D}| \ll 2^\eta$, inequality (3) can be rewritten as
$$p > n \cdot 2^{(s_2+\eta)(|\mathcal{D}|-1)}$$
or
$$|p| \ge \lceil \log_2 n \rceil + (s_2 + \eta)(|\mathcal{D}| - 1).$$
In addition, as our scheme relies on the Hidden Modular Group Order assumption, a security parameter s_1 has to be added to the length of p, see Trostle and Parrish [31] for more details. Finally, p is a prime of length
$$|p| = s_1 + \lceil \log_2 n \rceil + (s_2 + \eta)(|\mathcal{D}| - 1).$$

How to select η

To enable decryption of encrypted monomials Σ_j, the following additional condition is required for η:
$$2^\eta > \Sigma_j = \sum_{i=1}^{n} (R_i.c)^j$$
for all values of $R_i.c, j$. Therewith,
$$2^\eta > n \cdot (|\mathcal{D}| - 1)^{|\mathcal{D}|-1}$$
or
$$\eta = \lceil \log_2 n + (|\mathcal{D}| - 1) \cdot \log_2(|\mathcal{D}| - 1) \rceil.$$
We will formally prove IND-CPA for this encryption scheme in Section III-D.

C. Detailed Protocol Description

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

1) $\textsc{KeyGen}(s)$: Based on security parameters 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 $\textsc{KeyGen}(s_1, s_2, n, |\mathcal{D}|)$ for the somewhat homomorphic encryption, determining an upper bound n for the total number of patient records that might be stored and value for the domain $\mathcal{D}$ of the countable field. The secret key $\mathcal{S}$ is the output of the somewhat homomorphic encryption and K, i.e., $\mathcal{S} := \{p, b, K\}$.

2) $\textsc{Encrypt}(\mathcal{S}, \mathcal{R})$: 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., $\textsc{Enc}(\{p, b\}, R_i.c)$.
- 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

$$E_{R_i} := \{\textsc{Enc}(\{p, b\}, R_i.c), IV, \text{AES}_K - \text{CBC}(R_i)\}.$$

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

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

4) $\textsc{PrepareQuery}(\mathcal{S}, \chi)$: 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-A. To increase performance, the coefficients a_j are not sent to the cloud. The cloud will be only required to compute the monomials. Storing a_j's locally not only reduces computation overhead on the cloud, but will also perfectly prevent the cloud from learning which value is counted. Consequently in $EPiC$, the output Q of $\textsc{PrepareQuery}$ that is sent to the cloud is empty.

5) $\textsc{ProcessQuery}(Q, \mathcal{E})$: Based on the data set size and the cloud configuration, the MapReduce framework has selected M Mapper nodes and R Reducer nodes during $\textsc{Upload}$. 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 1 to $|\mathcal{D}| - 1$) of the countable field and output key-value pairs for each exponent which contain the order and value of the computed exponent. In MapReduce, output of the Mappers is then automatically sent to Reducers ("emit"). Each Reducer, based on the given pairs of order and value of the exponents, computes the sum of exponents of the same order and sends the results back to the cloud user. In the reduce phase, the size of those results is much smaller than the original set size, therefore allowing the cloud user to quickly compute the final result locally.

Algorithm 4: $\textsc{ProcessQuery}$

```
MapReduce Framework:
  select $M$ Mappers and $R$ Reducers
For each Mapper $m$:
  while input available do
      read $\{\textsc{Enc}(\{p,b\}, R_i.c), IV, \text{AES}_K - \text{CBC}(R_i)\}$
      for $j = 1$ to $|\mathcal{D}| - 1$ do
          emit $\{j, (\textsc{Enc}(\{p,b\}, R_i.c))^j\}$
      end
  end
For each Reducer $r$:
  for $j = 1$ to $|\mathcal{D}| - 1$ do
      $\Sigma_{rj} \leftarrow 0$
  end
  while input available do
      read $\{j, (\textsc{Enc}(R_i.c))^j\}$
      $\Sigma_{rj} \leftarrow \Sigma_{rj} + (\textsc{Enc}(R_i.c))^j$
  end
  for $j = 1$ to $|\mathcal{D}| - 1$ do
      write $\{j, \Sigma_{rj}\}$
  end
```

6) $\textsc{Decode}(\mathcal{S}, E_\Sigma)$: Cloud user $\mathcal{U}$ receives E_Σ, the sequence of encrypted sums. As of Section III-A, $\mathcal{U}$ computes $\Sigma_j := \textsc{Dec}(\{p, b\}, E_{\Sigma_j}), 1 \leq j \leq |\mathcal{D}|$ as shown in Algorithm 5.

Using the Σ_j and the previously computed a_j, $\mathcal{U}$ computes the total number of occurrences of χ in the outsourced set of records $\sigma := \sum_{j=0}^{|\mathcal{D}|-1} a_j \cdot \Sigma_j$.

Algorithm 5: $\textsc{Decode}$

```
$\mathcal{U}$:
  while input available do
      read $\{j, \Sigma_{rj}\}$;
      $\Sigma_j \leftarrow \Sigma_j + \Sigma_{rj}$;
  end
  $\sigma \leftarrow \sum_{j=0}^{|\mathcal{D}|-1} a_j \cdot \Sigma_j$
```

D. 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 [31]. 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 η least significant bits.

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

$$Pr[\mathcal{A} \text{ predicts } LSB(e) = m] \leq \frac{1}{2^\eta} + \epsilon(s) \qquad (4)$$

for any plaintext m, and ciphertext e, where $LSB(e)$ are the η least significant plaintext bits corresponding to ciphertext e, 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 bitstring m. $\mathcal{O}$ randomly selects $b \in \{0,1\}$. If $b = 0$, then the oracle randomly changes (at least one of) the low order bits of m and encrypts the result to e using Trostle and Parrish's scheme. If $b = 1$, the oracle sends the encryption e of m back to $\mathcal{A}$. Now, $\mathcal{A}$ has to decide whether $LSB(e) = m$.

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 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 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 e to $\mathcal{A}$. This value e is again simply forwarded to $\mathcal{A}'$. If $\mathcal{A}'$ outputs a value $b' = b$, then $\mathcal{A}$ knows that e corresponds to (unmodified) m_b and outputs $LSB(e) = m$. Otherwise, if $\mathcal{A}'$ aborts, $\mathcal{A}$ 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 the encryption.

Our notion of privacy-preserving for cloud-based counting security is formalized in Definition 2. We need to prove two properties, the first corresponds to storage-privacy, and the second corresponds to counting-privacy. We consider the specific case of the detailed protocol described in Section III-C.

- **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
  $$Pr(\text{Game}_1(\mathcal{A}) = 1) \leq \frac{1}{2} + \epsilon_1(s) \qquad (5)$$
  is violated.
  However, note that Q_b is empty and leaks no information. E_{Σ_b} can be computed by anyone who has access to $\mathcal{E}_b$ and therefore does not leak any additional information about b than $\mathcal{E}_b$. This is because of the specific somewhat homomorphic properties of EPiC encryption. Therefore, only $\mathcal{E}_b$ can help $\mathcal{A}$ in guessing the value of b. However, E_b is the result of encrypting $\mathcal{R}_b$. Therefore, the storage-privacy property reduces to the IND-CPA property of EPiC's encryption, where $\mathcal{R}_0$ and $\mathcal{R}_1$ represent the selected plaintexts and $\mathcal{E}_b$ the challenge ciphertext. The existence of a PPT adversary who can guess bit b with a non-negligible probability would imply that EPiC is not IND-CPA secure.
- **Counting-privacy:** The counting-privacy property is defined in the challenge phase of Game_2 as formalized in Algorithm 3.
  No PPT adversary should be able to guess bit b with probability higher than $\frac{1}{2} + \epsilon_2(s)$ for some negligible function $\epsilon_2(s)$, i.e.,
  $$Pr(\text{Game}_2(\mathcal{A}) = 1) \leq \frac{1}{2} + \epsilon_2(s). \qquad (6)$$
  Here, b determines the evaluation point χ_b for forming the query Q_b. A powerful property of our protocol is that no information about χ_b (or b) is sent to the cloud. As a matter of fact, the output Q of PrepareQuery that is sent to the cloud is always empty. The user receives from the cloud the output of $\text{ProcessQuery}(\mathcal{S})$ which consists of the sum of monomials evaluations $\sum_{i=1}^{n} (\text{Enc}(R_i.c))^j$ for every j. The user can then compute the answer to $\sum_{i=1}^{n} P_\chi(x)$ for any χ. The cloud computation, and the communicated information being completely independent from the value b, the probability that an adversary guesses b is exactly $\frac{1}{2}$. From the perspective of counting-privacy, the proposed scheme is even information-theoretically ("unconditionally") secure.

■

IV. Evaluation

To show its real-world applicability, we have implemented and evaluated EPiC in the Hadoop MapReduce framework v1.0.3 [4]. The source code is available for download [3].

Our evaluation is twofold: first, we have evaluated EPiC on a local "cloud" comprising just a single server with multiple CPUs. Moreover, we have deployed and evaluated EPiC on Amazon's public MapReduce cloud [1].

Implementation: Our EPiC implementation is written in Java, and all cryptographic operations are *unoptimized*, relying on Java's standard BigNumber data type. Still, exponentiation, $\textsc{Exponentiate}(\mathcal{C}, j)$, e.g., 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 < 1ms. 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. [20] – 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 [31] for good security and $s_2 = |r| = 160$ bits. We have implemented a data generator program to randomly generate patient records with a countable field. We set $|\mathcal{D}| = 16$, allowing for 16 different categories or types of diseases. As our evaluation targets comparing the performance of EPiC counting to non-private counting, we use encryption and decryption on the countable field only. Other fields are not considered during the evaluation.

IO overhead: As shown in Algorithm 4, each Mapper writes $|\mathcal{D}| - 1$ exponents for each record to the Reducers. Although the Mappers and the Reducers exchange data usually via high-speed links, the data transfer time is not negligible and can be a key aspect of performance [9]. In the context of our EPiC protocol, we reduce the exchanged data among Mappers and Reducers by letting the Mappers compute the sum of exponents of the same order before giving the results to the Reducers. In other words, the sum computation is not only performed at the Reducers, but also at the Mappers. This improvement is shown in Algorithm 6.

Algorithm 6: Improved Mapper

```
For each Mapper m:
  for $j = 1$ to $|\mathcal{D}| - 1$ do
      $\Sigma_{mj} \leftarrow 0$
  end
  while input available do
      read $\{\textsc{Enc}(\{p, b\}, R_i.c), IV, \text{AES}_K - \text{CBC}(R_i)\}$
      for $j = 1$ to $|\mathcal{D}| - 1$ do
          $\Sigma_{mj} \leftarrow \Sigma_{mj} + (\textsc{Enc}(R_i.c))^j$
      end
  end
  for $j = 1$ to $|\mathcal{D}| - 1$ do
      emit $\{j, \Sigma_{mj}\}$
  end
```

A. Local Evaluation

First, we have evaluated EPiC's performance for small-scale data sets on a server "cloud". The server, running Arch Linux 2011.08.19, is equipped with 2 Intel Xeon E5620 2.4GHz processors, each with 4 cores and 12MB of cache. This reflects to a cloud with a total 8 nodes (Mappers/Reducers). Total memory of the system is 48GB.

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 10GB to 100GB. Figure 1 shows the average counting time of the whole data set for different sizes. Each sampling point is measured for 10 runs, relative standard deviation was low at $\approx 10\%$. 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]. We show both, the time for EPiC as well as the time a "non-privacy-preserving" counting takes, i.e., the countable field is not encrypted and directly counted.

Moreover, we also show the overhead ratio between EPiC and non-private counting. The additional overhead introduced by EPiC over non-private counting is less than 36%. As Amazon's pricing scales directly proportional with the CPU time. This would also the additional amount of money user $\mathcal{U}$ would have to pay Amazon. We conjecture that only 36% overhead over non-privacy-preserving counting is acceptable in many real-world situations, rendering EPiC practical.

Note that the overhead suddenly increases from around 50GB due to the fact that the system has 48GB of RAM, and more processed data need more caching and IO operations, thus causing higher overhead. The overhead is also considerable in non-private counting (Figure 2).

2) Variable record size: To also evaluate the effect of the size of the records on general performance, we run the system with a fixed data set size of 30GB. The record size is then changed from 100KB to 1MB (accordingly, the number of records changes from 300000 to 30000). Figure 3 shows that, while IO time remains unchanged, a higher number of records increases counting time in EPiC. However, for small record sizes such as 100KB, the ratio between EPiC and non-private counting is still under 80%. That is, EPiC is efficient even for small patient records.

B. Amazon evaluation

To also evaluate the scalability of EPiC with big data, we conducted intensive experiments on Amazon's public cloud. As Amazon imposes an (initial) limit of 20 instances per job, we have restricted ourselves to 20 Extra Large On-Demand instances [2]. Each instance comprises 8 processors with 2.13GHz Intel Xeon CPU, 4MB of cache, and total 7GB of memory. The comparison of performance between EPiC and non-private counting is depicted in Figure 4. Again, the overhead remains modest and remains below 60% even for huge total data sizes of 1TByte. We conclude that EPiC is practical in many real-world scenarios.

V. Related Work

Protection 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

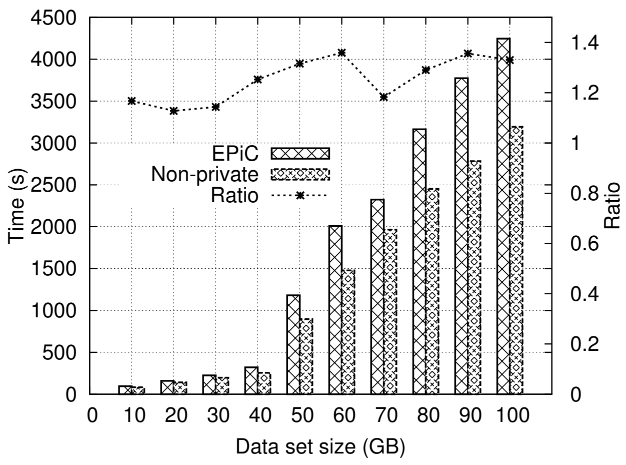

Fig. 1. Local evaluation, total time

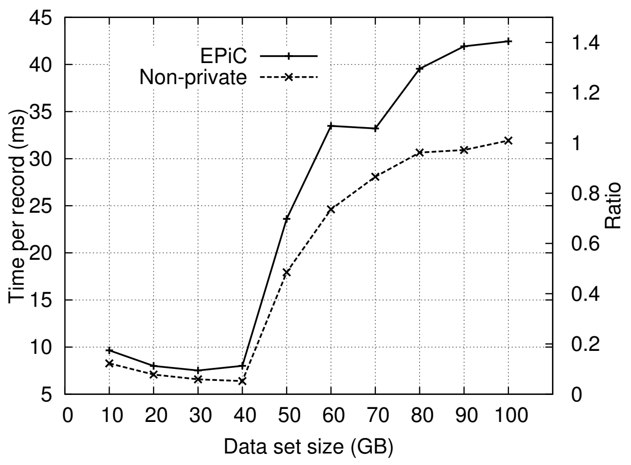

Fig. 2. Local evaluation, time per record

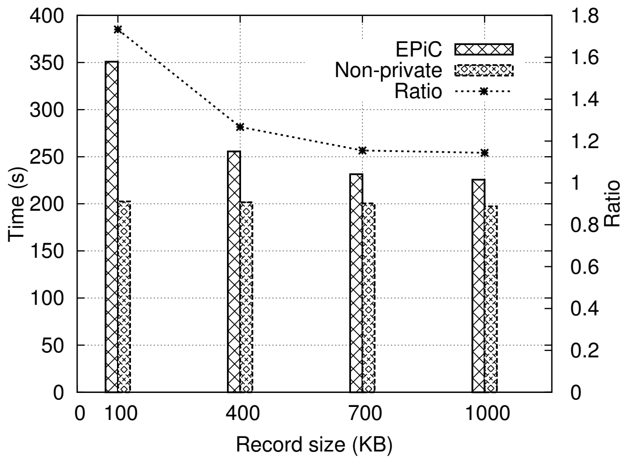

Fig. 3. Local evaluation, 30GB data sets, varying record size

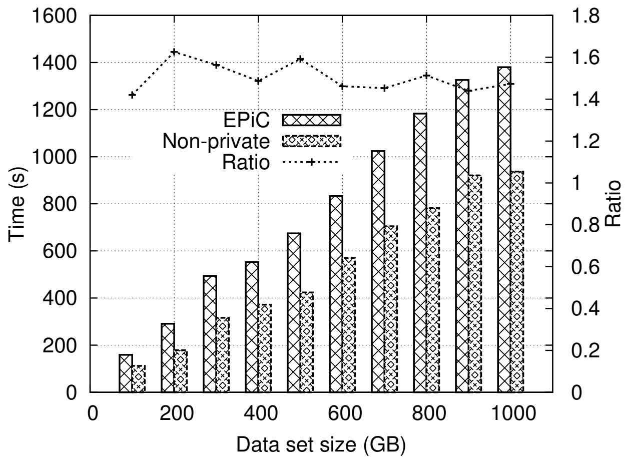

Fig. 4. Amazon evaluation, total time

(see Lipmaa [21] for an overview), current implementations indicate high storage and computational overhead [12, 24]. This renders fully homomorphic encryption impractical for cloud computing.

Similar to EPiC, Lauter et al. [20] observe that, depending on the application, weaker "somewhat" homomorphic encryption might be sufficient. Lauter et al. [20]'s scheme is based on a protocol for lattice-based cryptography by Brakerski and Vaikuntanathan [8]. 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 [19]. 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 [7]). 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 [31] 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 [6, 23, 28, 32]. 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. [6] 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 [10]. 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 [27]. EPiC, however, addresses the opposite problem where a user does not trust the cloud infrastructure.

VI. Conclusion

In this paper, we presented EPiC to address a fundamental problem of statistics computation on outsourced data: privacy-preserving counting. EPiC's main idea is to count occurrences of patterns in outsourced data by a privacy-preserving evaluation of special polynomials. Using a "somewhat homomorphic" encryption mechanism, the cloud does neither learn any information about outsourced data nor about 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] Anonymized. EPiC Source Code, 2012.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

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

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

[17] 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.

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

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

[20] 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.

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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 Environments, Hong Kong, China, 2011. http://www.gsd.inesc-id.pt/~mpc/pubs/diamonds-final.pdf.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

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