Secure Federated Aggregate-Count Queries on Medical Patient Databases Using Fully-Homomorphic Cryptography

Alexander T. Leighton[1]

Yun William Yu[2*]

[1]Harvard Medical School, Department of Biomedical Informatics, Boston, MA, USA
[2]University of Toronto, Department of Mathematics, Toronto, Ontario, Canada

Abstract

Electronic health records (EHR) are often siloed across a network of hospitals, but researchers may wish to perform aggregate count queries on said records in entirety—e.g. How many patients have diabetes? Prior work has established a strong approach to answering these queries in the form of probabilistic sketching algorithms like LogLog and HyperLogLog; however, it has remained somewhat of an open question how these algorithms should be made truly *private*. While many works in the computational biology community—as well as the computer science community at large—have attempted to solve this problem using differential privacy, these methods involve adding noise and still reveal some amount of non-trivial information. Here, we prototype a new protocol using fully homomorphic encryption that is trivially secured even in the setting of quantum-capable adversaries, as it reveals no information other than that which can be trivially gained from final numerical estimation. Simulating up to 16 parties on a single CPU thread takes no longer than 20 minutes to return an estimate with expected 6% approximation error; furthermore, the protocol is parallelizable across both parties and cores, so, in practice, with optimized code, we might expect sub-minute processing time for each party.

Key Words: Privacy, Fully Homomorphic Encryption, LogLog algorithm

Availibility: Our software is available at `https://github.com/atleighton/rlwe-hll`

*Corresponding Author. Email: ywyu@math.toronto.edu

1 Introduction

We propose a Fully-Homomorphic (FH), Ring-Learning With Errors (RLWE) based protocol to the federated multiset cardinality estimate problem, revealing negligible information other than that trivially derived from the approximate count. The multiset cardinality estimate problem attempts to answer the question of the cardinality of the union of a number of sets; in other words, how many unique data-points are there in a collection of datasets? This problem has been studied in a variety of contexts with many different constraints. Here, we are consider the medical context of how many unique patients in a hospital-network match a given query criteria.

The federated multiset cardinality estimate problem is another name for the primitive clinical query, which asks how many patient records match some condition [1]. As such, it is pertinent to medical research and the protection of biomedical data. The problem, though seemingly trivial in nature, is actually quite deep: For example, taking a naive sum of counts from each hospital in a network risks overcounting patients with records at multiple hospitals [2]. Alternately, centralizing all of the information using hashed identifiers [3] comes at the cost of patient privacy: the central party learns about the health status of all patients across the entire network. Thus, we take it as a given a trade-off exists between accuracy and privacy when answering such queries [4]. It is established that probabilistic cardinality estimators are a good way of addressing this problem [5].

Privacy of cardinality estimation is of increasing interest, both in- and outside of medical informatics. Importantly, an aptly title 2018 paper argues "Cardinality Estimators do not Preserve Privacy" [6], demonstrating that an adversary with access to all intermediate HLL representations during computation can reconstruct a large fraction of records. When only some intermediate HLL representations are given, that does help protect privacy—the privacy of naive HyperLogLog sketches under the context of 10-anonymity [7] was explored in an ISMB2021 paper [8]. Furthermore, hiding individual hospital sketches by combining them using partially homomorphic encryption, but then revealing the combined sketch, was analyzed in the trade-offs paper above [5]; however, as that paper showed, the revealed combined sketch still leaks a significant amount of patient information, though less than individual hospital sketch would. A 2020 paper explores the use of Bloom filters and differential privacy (Figure 1) to decrease the information leakage of HyperLogLog (HLL) sketches among multiple untrusted parties [9]. Meanwhile, an aptly title 2018 paper argues "Cardinality Estimators do not Preserve Privacy" [6], demonstrating that an adversary with access to intermediate HLL representations during computation can reconstruct a large fraction of records. These results all add noise achieve privacy before sending data and revealing intermediate plaintext sketches; we demonstrate a cleaner solution.

Our solution begins with the notion, such plaintext sketches need not be revealed at all; Gentry's landmark 2009 work demonstrating the existence of fully homomorphic encryption (FHE) schemes proved FHE a theoretically viable method for securely tackling nearly any problem in computation [10]. But, despite being remarkably efficient in its theoretical context, the runtime and communication complexity of Gentry's multiparty computation prevented it from being immediately applied to real-world problems. In the years after, many papers built on this original work. On the cryptographic theory side, alternate FHE protocols with better complexity were developed, as well as extensions of the key-generation protocols to allow for multi-party encryption and decryption [11, 12, 13, 14]. Practical implementations of RLWE based FHE protocols are just beginning to become available, yet they have already been utilized in the computational biology community to craft secure versions of existing algorithm often run on sensitive data, most notably secure GWAS in a 2020 PNAS article [15], as well as a more recent 2021 Nature Communications

paper [16].

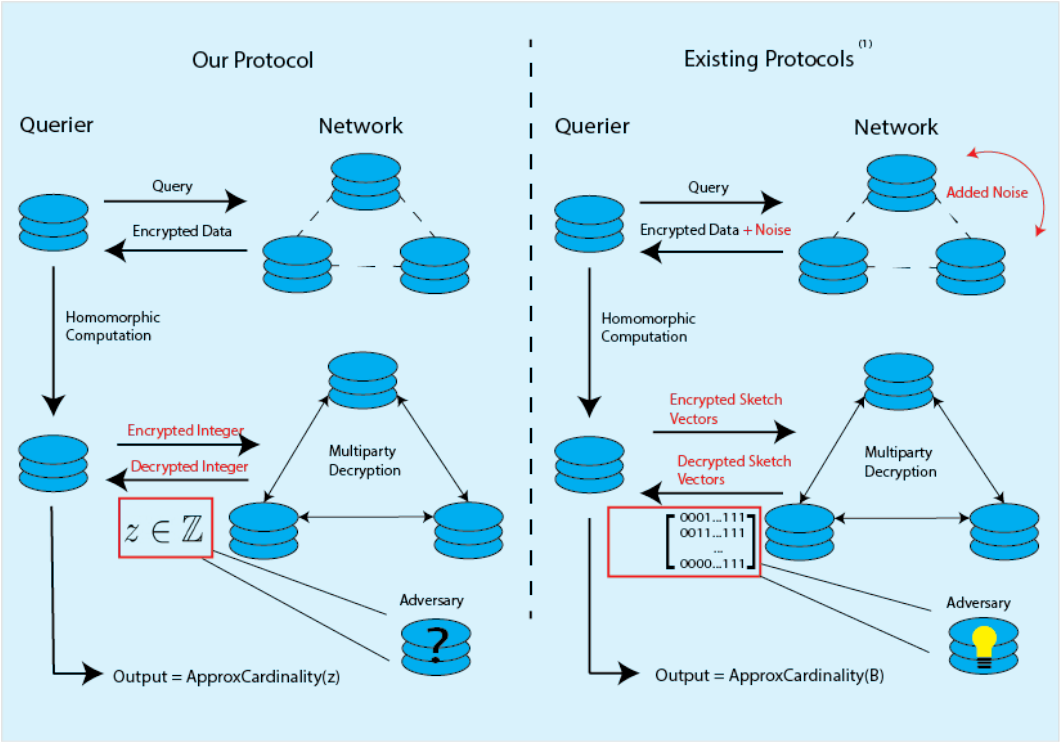

Figure 1: Our protocol differs fundamentally from protocols proposed in Kreuter, et al [9], Yu, Weber [5], and Tao, Weber, Yu [8]. These protocols reveal intermediate sketch vectors to all compute parties: Our protocol reveals only an integer, reducing accuracy by a small fraction that may be evened out by using ~ 1.5 times as many buckets.

Even the fast GWAS protocols require hours of total single-threaded compute [15], reducible to tens of minutes using parallelization. This is suitable for complicated analyses, but not as much in the workflow for clinical query systems, which are expected to return queries in less than a minute. But, of course, a decade ago, such processes would likely have taken even longer without any additional privacy. We demonstrate that sub-minute runtimes for the multi-set cardinality estimation are currently possible using the residue number system (RNS) [17] variant of the BFV encryption scheme [18] as implemented in the Palisade somtware library [19]. **Our key conceptual advance is recognizing we could efficiently combine sketches from the LogLog [20] sketching algorithm within the BFVrns encryption framework, revealing only the algorithm's output and no intermediate values.**

2 Methods

Here we approximate the number of unique data points in a distributed dataset matching a given query using privacy-preserving FHE. We provide simulation code and bench-marking for the honest-but-curious (HBC) cryptographic setting and describe how the algorithm can be easily modified to be secure in the malicious setting. Our implementation uses the Palisade library's BFVrns [17]

framework, an implementation the Residue Number System (RNS) variant of the Brakerski, Fan, Vercauteren (BFV) [18] scale-invariant homomorphic encryption scheme.[1] Using this framework, we modify the **LogLog** sketching algorithm [20] such that all *information-revealing* computation takes place inside Palisade's BFVrns framework on ciphertext. As such, no party gains information about any other party's underlying data except for the final approximate count-query estimate from the LogLog sketch.

2.1 Fully Homomorphic Encryption (FHE) & Ring Learning with Errors (RLWE)

A homomorphic encryption is an encryption scheme allowing some functions to be computed on cipher-texts without first decrypting. FHE extends to allowing *all* functions to be theoretically computed. In practice, this computation is a far more daunting task. An early solution to implementing this sort of arbitrary functionality between two parties, were Yao's **garbled-circuits**, but this approach is currently somewhat dated and is computationally far too expensive for most problems [21, 22]. Other approaches to the count-query use partially homomorphic public key schemes (such as ElGamal [23]), which only allow modular multiplications of cipher-texts [5, 9, 24]. However, these schemes, while relatively computationally efficient, suffer the drawback of not being able to perform additions inside the cryptographic framework. Either some computation has to be done in plaintext, resulting in information leakage [5, 9], or the protocol requires many rounds of back-and-forth communication [24], which is impractical across a large network with the possibility of dropout. We note that in the 2-party setting, there are extremely efficient exact count-query protocols based on oblivious transfer and symmetric-key encryption, but these are not easily extensible to the multi-party case [25]. In our paper, we expand upon the work of [5], using FHE to obviate the need to reveal an unencrypted sketch at any point in the protocol.

2.2 LogLog & HyperLogLog Sketching

Our protocol, along with most the work we've cited tackling the privatized multi-set cardinality estimate problem relies on the **LogLog**-family of multi-set cardinality estimators. The basic idea behind these estimators is given n uniformly random numbers between 0 and 1, the expected minimum of these numbers is $\frac{1}{n+1}$. This allows us to fairly quickly see we might estimate cardinality by hashing elements in an arbitrary set to uniformly random numbers between 0 and 1, and then derive an approximate cardinality by taking the inverse of the minimum such value. Assuming we are consistent with our choice of hash function, we get de-duplication (duplicate elements hash to the same value). This procedure is known as the **k-minimum values (KMV)** algorithm [26]—readers may notice that the generated sketch is also equivalent to the **MinHash** [27] sketch for Jaccard index, but the estimation is performed differently. As with **MinHash**, one can easily combine sketches of two sets A and B to return the sketch of the union $A \cup B$, because that simply requires taking the smaller of the hashed elements across both sketches.

The **LogLog** family of algorithms relies on the same intuition as **KMV**, but reformulates the procedure to both reduce space-complexity and variance [20]. First, instead of storing the minimum hash value exactly, one only needs to store the order-of-magnitude of the minimum hash value, giving rise to the "LogLog" name; storing the magnitude of the hashes to the nearest power of 2 provides an extra logarithmic reduction in size of the hash values themselves, which

[1] https://palisade-crypto.org [19]

are likewise logarithmic in the size of the set through simple binary encoding. Importantly, the double logarithmic space compression allows us to later use an exponential blow-up through a unary encoding of bit vectors while still maintaining reasonable logarithmic space.

Another key idea is that keeping track of multiple minimum hash values for a set of buckets reduces variance. In the **LogLog** algorithm, a fixed number of buckets, m is picked. Then, elements are assigned a random hash value, like in **KMV**, but also a random bucket. This is done trivially by allocating a certain number of bits of the hashed value to determine the bucket. We keep track of the minimum elements of each bucket, taking the combined minimum per bucket when merging into an aggregate approximation. Much of the rich historic work in the development of the **LogLog** family has aimed to construct lower-variance estimators given a collection of minimum elements within each bucket. The original LogLog estimator uses an arithmetic mean of the bucket values and had a standard error of around $\frac{1.30}{\sqrt{m}}$. SuperLogLog improves upon LogLog by truncating extreme data, getting an effect error of around $\frac{1.05}{\sqrt{m}}$, but at the cost of difficulty in analysis. The most commonly used today, HyperLogLog [28], replaces the arithmetic mean of LogLog with a harmonic mean, achieving a standard error of $\frac{1.04}{\sqrt{m}}$, better than SuperLogLog and without using conditional truncation; however, this is still within a constant factor of the foundational result. There are more recent advances in cardinality sketching, particularly in the theory literature. These include sketches that only require a constant number of bits per bucket [29], but HyperLogLog remains the most widely used of these methods in practice [30].

We present these alternative methods to give a fuller picture of cardinality estimation sketches. However, in this work, we make the deliberate choice to use the older LogLog sketch instead of the more commonly used ones, as we've found it allows the sketches to be merged efficiently while still encrypted and is only less accurate by a small constant factor in the number of buckets. Furthermore, we are not aware of a good method for computing the exponents required of newer HyperLogLog sketches in the encrypted, non-information-leaking setting, or of doing the conditional truncation used by SuperLogLog. But, by leveraging the simpler LogLog sketch, we are able to encode the estimation procedure entirely into the encrypted homomorphic computations, resolving the issues earlier works had with plaintext information leakage when using HyperLogLog based procedures.

2.3 Algorithms

Our Protocol is comprised of four algorithms. ComputeSketches takes as input the query and returns m unary vectors representing a party's sketches of the query. EncryptSketches, as the name would suggest, encrypts the unary sketch vectors using a multiparty public key we assume was generated during preprocessing. mergeSketches multiplies the encrypted sketches belonging to the same buckets, resulting in one unified, encrypted, minimal sketch for each bucket. It then adds up and outputs the total number of 1's in the resulting unary vectors, still in ciphertext. DecryptSketches simply performs the multiparty decryption of the output value.

Importantly, the unary encoding allows us to represent maximum as a vector-wise multiplication—if $5 = 000001$, and $2 = 001111$ (counting the number of 0's), then $\max(5, 2) = 000001 \times 001111 = 000001 = 5$. Furthermore, we can convert that vector back into an integer by subtracting component wise from 1, and the summing—i.e. $000001 \rightarrow 111110 \rightarrow 5$, all while remaining in ciphertext

and using just additions and multiplications.

Algorithm 1: ComputeSketches

Data: $QUERY$, m, $DATA_i$; // $m = NUM_BUCKETS$, $DATA_i$= Party i's data
Result: $s_{i,j}$; // Party i's sketches of all buckets j
for $j \in [1:m]$ **do**
  $s_{i,j} = 1$
for $p \in DATA_i$ **do**
  if $QUERY(p) == TRUE$ **then**
    $b = 128BitSha1Hash(identifier(p))$; // We use patient SSN as $identifier$
    $BUCKET \equiv b[1:64] \pmod{m}$;
    $Sketch_j =$ **index of first 1 in** $b[65:128]$; // $Sketch_j \in [1..64]$
    ;
    if $Sketch_j > s_{i,BUCKET}$ **then**
      $s_{i,BUCKET} = Sketch_j$;

Algorithm 2: EncryptSketches

Data: $s_{i,j}$; // Party i's sketches of all buckets j
Result: $c_{i,j}$; // ciphertexts of encrypted 32-bit plaintext unary vectors from sketches
for $j \in [1:]$ **do**
  $p_{i,j} = 0^{s_{i,j}}|1^{32-s_{i,j}}$; // $p_{i,j}$ is a 32-bit plaintext unary encoding of party i's sketches
  $c_{i,j} =$ **Enc**$(p_{i,j})$

Algorithm 3: mergeSketches

Data: $\forall j \in [1:m], \forall i \in [1:n], c_{i,j}$; // with n parties and m buckets
Result: c_N ; // ciphertext of integer output used in estimator
for $j \in [1:m]$ **do**
  $c_{j,mult} = c_{1,j} \circ c_{2,j} \circ ... \circ c_{n,j}$

$$c_{mult} = \sum_{j \in m} c_{j,mult}$$

$$c_N = \sum_{k=1}^{32} c_{mult}[k]$$

/* Note: here $\circ$ is the element wise product of the ciphertext vectors. */
/* Our implementation computes $c_{j,mult}$ using binary trees to reduce multiplicative depth */

Algorithm 4: DecryptSketches

Data: c_N ; // Ciphertext of integer output used in estimator
Result: N ; // Plaintext integer used to estimate cardinality
$partial_1 = Partial$Decrypt(c_N, sk_1)
for $i \in [2:n-1]$ **do**
  $partial_i = Partial$Decrypt$(partial_{i-1}, sk_i)$
$N = Partial$Decrypt$(partial_{n-1}, sk_n)$

We estimate cardinality following the regular **loglog** algorithm estimation in plaintext [20]:

$$E \coloneqq a_m m 2^{\frac{33m-N}{m}} \tag{1}$$

where,

$$a_m \coloneqq \left(\Gamma(-1/m)\frac{1-2^{1/m}}{\ln(2)}\right)^{-m}$$

and,

$$\Gamma(s) \coloneqq \frac{1}{s}\int_0^\infty e^{-t}t^s dt$$

Since the number of buckets m is known and must be agreed upon to enact the protocol, the only sensitive information is in that formula is N. The information revealed by E is thus equivalent to the information revealed by N.

3 Results and Discussion

3.1 Runtime

While FHE is computationally expensive, with the optimization of libraries like PALISADE and decreasing cost of computing resources, FHE-based solutions are becoming increasingly practical. For the multi-set cardinality estimation problem in particular, algorithms like LogLog and HyperLogLog reduce the complexity by double log factors, allowing some room to apply FHE. To this end, we wrote some benchmarking code to determine the single-threaded runtimes of the cryptographic steps within our protocol, using a 128-bit security parameter (HEStd_128_classic), and varying the number of buckets and parties. Our benchmarking code is available on Github at `https://github.com/atleighton/rlwe-hll`.

We note that in the semi-honest setting, there is no need for duplication of computation, so each of the algorithms can be done sequentially. As expected, we found that the computation time is dominated by MergeSketches. We further divide the runtime of the MergeSketches step into two parts: multiplication and addition. In the multiplication step, the combined sketch among all parties is computed for each bucket. In the addition step, we sum the combined unary sketches for each bucket and then sum these sums to get the total number of ones in the merged sketches. This number allows us to compute the arithmetic mean of the LogLog sketch.

The multiplication step's runtime (Figure 2) is linear in the number of buckets and nonlinear in the number of parties. The increase in runtime is linear in the number of parties from 2 to 4 to 8, but then suddenly jumps when the number of parties goes to 16. We believe this to be primarily a function of the multiplicative depth of the overall computation, combined with 'relinearization' cost of the cryptosystem. RLWE-based cryptosystems are fully homomorphic in that they can perform both addition and multiplication, but multiplication is more expensive; it is expensive because each multiplication adds 'noise' to the ciphertext, and when that noise builds up beyond a certain threshold, a relinearization operation is needed to denoise it. To be clear this 'noise' is different from the noise in privacy-based approaches and can be though of as a variable use to perform encrypted computation on exact values. In order to multiply together the n sketches from each party, our algorithm needs to perform $n-1$ multiplications; however, we can arrange the multiplication as a binary tree, multiplying together the first two sketches, then the next two, etc. Using this binary

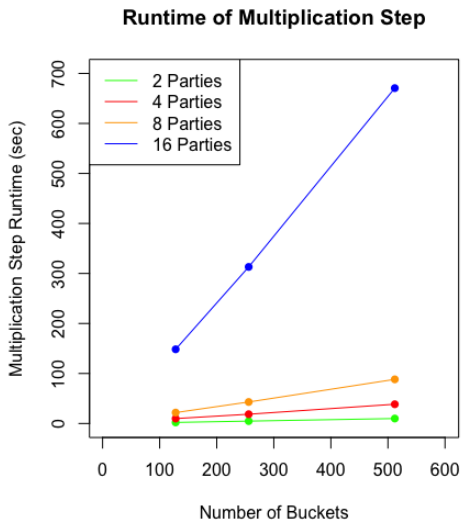

Figure 2: Runtime of multiplication step of MERGESketches. $\Theta(n \cdot m)$ calls to homomorphic vector multiply are made for n parties and m buckets. However, repeated multiplications of the same ciphertext are not constant time due to the relinearization needed in RLWE, so even though the number of calls is linear in m, the runtime is not.

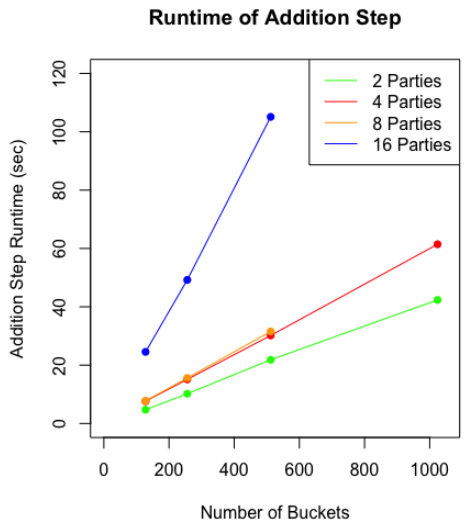

Figure 3: Runtime of combined addition step of MERGESketches. $\Theta(n \cdot m)$ calls to homomorphic vector add are made for n buckets and m parties. Note: We are unsure why run times are nearly identical for $n = 4$ and $n = 8$ parties, but, after thorough testing, we suspect an idiosyncrasy in the PALISADE software library.

tree, the 'multiplicative depth,' the number of multiplications that any one ciphertext has to go through, is only $\log(n)$. It appears that in our benchmarking, when the multiplicative depth goes from 3 (for 8 parties) to 4 (for 16 parties), we are triggering an expensive relinearization operation.

The addition step again is linear in the number of buckets and nonlinear in the number of parties (Figure 3). Indeed, it seems that the runtime for 4 and 8 parties is nearly identical, but suddenly jumps when going to 16 parties. The nonlinearity in the number of parties is a bit more surprising, and we do not fully understand why it is present. After thorough testing, we suspect an idiosyncrasy in the PALISADE software library.

However, note that simulating up to 16 parties on a single CPU thread still takes less than 20 minutes to return an estimate using 512 buckets, for an expected 6% error. With parallelization across computing parties and cores in the semi-honest setting, it is likely feasible to return queries with less than 1 minute of compute per party.

3.2 Security

In this section, we argue that our protocol is secure, as long as the well-known BFVrns is secure also. We provide a proof-sketch our protocol's *privacy* as we've defined it reduces to the security of the underlying BFVrns encryption scheme in the semi-honest setting via simulation [31] . For convenience, we separate the roles of the computing and data parties, and argue the security of our protocol against malicious computing parties (though not necessarily against data parties). We

make this distinction, so we don't have to worry about malicious data parties simply choosing to send empty sets, learning the union cardinality of the remaining honest party or parties' sets.

3.2.1 Simplified Algorithm

Let us first give a concrete simplified version of the algorithm, separating out the role of the computing and data parties, but having all messages between parties as public broadcasts to the network to simplify the analysis. We will also assume that communication between parties are secure and verified through standard public-private key encryption.

1. Prior to execution, the computing parties $C_1, \ldots, C_n$ perform a joint key generation using Palisade multiparty BFVrns, so there exists a joint encryption public key K and individual partial decryption private keys S_i.

2. Each data party $D_1, \ldots, D_p$ starts with an input set A_i. They compute and broadcast to the network $E_i = \textsc{Encrypt}_K(\textsc{Compute}Sketches(A_i))$.

3. Each computing party C_i computes $O_i = Partial\textsc{Decrypt}_{S_i}(\textsc{Merge}Sketches(E_1, \ldots, E_p))$ and broadcasts that to the network.

4. All parties then compute $O = \textsc{Decrypt}Sketches(O_1, \ldots, O_n)$, the plaintext integer output of the LogLog algorithm, which can be scaled by equation 1.

Of course, in the semi-honest setting, many of the computation steps can be parallelized across parties, because we can trust the parties to perform the steps correctly, but this simplified (slower) algorithm is both easier to analyze, and also more secure in the malicious setting.

3.2.2 Semi-honest network

Here we argue our protocols security reduces to the security of Palisade's implementation of the BFVrns cryptosystem. Consider the view of a semi-honest adversary who has corrupted all parties except computing party C_1 and data party D_1. The adversary thus knows all of the sets except A_1, all of the secret keys except S_1, all of the public broadcasts, which are computationally indistinguishable except insofar as they allow the computation of the output O. The real view of the adversary is therefore $\{A_2, \ldots, A_p, E_1, \ldots, E_p, K, S_2, \ldots, S_n, O_1, \ldots, O_n, O\}$.

Let the functionality $f(X)$ be the LogLog integer used to estimate cardinality (i.e. the sum of all the buckets in a LogLog sketch of the set X). That is to say, $f(A_1 \cup A_2 \cup \cdots \cup A_p) = O$. This functionality is deterministic, as the random hash function in the LogLog sketch is predetermined.

The Simulator then starts by simulating an m-party keygen with a new private key $S_1' \neq S_1$ and using $S_i' = S_i$ for $i > 1$ so that it has access to a new public key K' based on this set of private keys (of course, the simulator does not have access to S_1). We can create a set $\tilde{A}$ such that $A_i \subseteq \tilde{A}$ for $i > 1$ and $f(\tilde{A}) = O$, which is doable in polynomial time by just augmenting by random new elements until the sum of the buckets is O, which will take approximately $O(|A_1|)$ time. Then define $A_1' = \tilde{A} - \cup_{i=2}^{m} A_i$. Then, the simulator lets $E_1' = \textsc{Encrypt}_{K'}(\textsc{Compute}Sketches(A_1'))$ and $E_i' = \textsc{Encrypt}_{K'}(\textsc{Compute}Sketches(A_i))$ for all $i > 1$. We can get simulated partial decryptions $O_i' = Partial\textsc{Decrypt}_{S_i'}(\textsc{Merge}Sketches(E_1', \ldots, E_p'))$, but note that by construction $O = \textsc{Decrypt}Sketches(O_1, \ldots, O_n)$. Then the final simulated view for the adversary is $\{A_2, \ldots, A_p, E_1', \ldots, E_p', K', S_2', \ldots, S_n', O_1', \ldots, O_n', O\}$.

The simulated view is computationally indistinguishable view for the adversary from the real view. Thus, in a semi-honest setting, even if all but one data party and all but one computing party are corrupted, the adversary can learn nothing more than the output O from the protocol. Note that the setting where all data parties are corrupted is of course uninteresting since the adversary already then has access to all of the underlying data, so this means that in the semi-honest setting, our protocol is secure so long as a single computing party is honest.

3.2.3 Malicious setting

In the malicious setting, data parties may try to learn information about the sets of the other honest data parties by sending adversarially crafted possibly malformed LogLog sketches; even simply choosing to send empty sets is sufficient to reveal the union cardinality of the honest parties' sets. However, so long as all data parties and at least one computing party are honest, our protocol is secure, contingent on the assumption that the distributed key generation can be made secure in the malicious setting. If any computing party sends an incorrect partial decryption, then the final decryption will simply fail (or more precisely, will give a random number). The malicious computing party can still use the correct partial decryption to learn the output O, which none of the other parties would in that case, but that is still within the privacy guarantee of the protocol. Additionally, broadcasting an incorrect partial decryption would reveal that the computing party is malicious. Thus, the malicious setting for compromised computing parties should reduce to the semi-honest setting above, and thus also be secure.

3.3 Accuracy

Our encrypted protocol for computing the LogLog sketch exactly matches the unencrypted equivalent, and therefore has the same accuracy. The standard error of **loglog** sketches is $\approx \frac{1.30}{\sqrt{\#\text{buckets}}}$ whereas the standard error of **HyperLogLog** approximations is $\approx \frac{1.04}{\sqrt{\#\text{buckets}}}$. It follows simply we are able to achieve similar accuracy, regardless our choice of sketching algorithms provided in the **loglog** case we increase the number of buckets by a factor of $\left(\frac{1.30}{1.04}\right)^2 = 1.56$.

4 Conclusion

In this paper, we prototype a novel approach to privately computing multiset cardinality estimates, using a RLWE-based FHE cryptosystem. We demonstrate our approach adds significant value to the current literature by fundamentally reducing the amount of sensitive-information leakage via fully homomorphic encryption and computing on encrypted, rather than plaintext, sketches. By keeping all intermediate computation in ciphertext, we ensure no information is revealed other than that which can be trivially derived from the resulting estimate (assuming security of established protocols and libraries cited). Furthermore, unlike many prior methods, our privacy guarantees do not depend on injecting noise into the intermediate computation, which would otherwise decrease accuracy. Hence, we question the need to even bother using noise-inducing privacy techniques on this problem at all. If additional differential privacy guarantees are desired, one could, homomorphically add noise to the encrypted output of the protocol. Still in this case, it remains advantageous to use our methodology.

To summarize, our work demonstrates that modern homomorphic encryption libraries are sufficiently fast in principle to be used in interactive federated queries, not just off-line analysis. We show leveraging FHE based MPC frameworks to be a sound plan-of-attack towards solving the problem of privately estimating federated multi-set cardinalities. As problems relating to the security and privacy of biomedical data grow in prevalence, we trust FHE frameworks will continue to simply and robustly solve a number of the less computationally-intensive privacy-based problems that arise going forward.

5 Acknowledgements

This work was supported by startup and bridge funding from the University of Toronto at Scarborough. We thank Isaac Kohane and Griffin Weber for introducing the federated count-query problem to us and for general guidance. Furthermore, we thank Adam Sealfon for fruitful discussions.

References

[1] Weber, G. M. Federated queries of clinical data repositories: scaling to a national network. *Journal of biomedical informatics* **55**, 231–236 (2015).

[2] Weber, G. M. Federated queries of clinical data repositories: the sum of the parts does not equal the whole. *Journal of the American Medical Informatics Association* **20**, e155–e161 (2013).

[3] Grannis, S. J., Overhage, J. M. & McDonald, C. J. Analysis of identifier performance using a deterministic linkage algorithm. In *Proceedings of the AMIA Symposium*, 305 (American Medical Informatics Association, 2002).

[4] Cho, H., Simmons, S., Kim, R. & Berger, B. Privacy-preserving biomedical database queries with optimal privacy-utility trade-offs. *Cell systems* **10**, 408–416 (2020).

[5] Yu, Y. W. & Weber, G. M. Balancing accuracy and privacy in federated queries of clinical data repositories: Algorithm development and validation. *J Med Internet Res* **22**, e18735 (2020). URL `https://www.jmir.org/2020/11/e18735`.

[6] Desfontaines, D., Lochbihler, A. & Basin, D. A. Cardinality estimators do not preserve privacy. *CoRR* **abs/1808.05879** (2018). URL `http://arxiv.org/abs/1808.05879`. `1808.05879`.

[7] Sweeney, L. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10**, 557–570 (2002).

[8] Tao, Z., Weber, G. M. & Yu, Y. W. Expected 10-anonymity of hyperloglog sketches for federated queries of clinical data repositories. *bioRxiv* (2021). URL `https://www.biorxiv.org/content/early/2021/02/01/2021.01.30.428918`. `https://www.biorxiv.org/content/early/2021/02/01/2021.01.30.428918.full.pdf`.

[9] Kreuter, B., Wright, C. W., Skvortsov, E. S., Mirisola, R. & Wang, Y. Privacy-preserving secure cardinality and frequency estimation. Tech. Rep., Google, LLC (2020).

[10] Gentry, C. A fully homomorphic encryption scheme (2009). `crypto.stanford.edu/craig`.

[11] Regev, O. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM* **56** (2009). URL `https://doi.org/10.1145/1568318.1568324`.

[12] Brakerski, Z. & Vaikuntanathan, V. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Rogaway, P. (ed.) *Advances in Cryptology – CRYPTO 2011*, 505–524 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2011).

[13] Brakerski, Z., Gentry, C. & Vaikuntanathan, V. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277 (2011). `https://ia.cr/2011/277`.

[14] Brakerski, Z., Gentry, C. & Vaikuntanathan, V. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory* **6** (2014). URL `https://doi.org/10.1145/2633600`.

[15] Blatt, M., Gusev, A., Polyakov, Y. & Goldwasser, S. Secure large-scale genome-wide association studies using homomorphic encryption. *PNAS* **117 (21)** (2020).

[16] Froelicher, D. *et al.* Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. *Nature Communications* **12** (2021). URL `https://doi.org/10.1038/s41467-021-25972-y`.

[17] Halevi, S., Polyakov, Y. & Shoup, V. An improved rns variant of the bfv homomorphic encryption scheme. *Cryptographers' Track at the RSA Conference* (2019). URL `https://eprint.iacr.org/2018/117.pdf`.

[18] Fan, J. & Vercauteren, F. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012). `https://ia.cr/2012/144`.

[19] Polyakov, Y., Rohloff, K., Ryan, G. W. & Cousins, D. Palisade lattice cryptography library. URL `https://palisade-crypto.org/`.

[20] Durand, M. & Flajolet, P. Loglog counting of large cardinalities. In Di Battista, G. & Zwick, U. (eds.) *Algorithms - ESA 2003*, 605–617 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2003).

[21] Yao, A. C.-C. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 162–167 (1986).

[22] Beaver, D., Micali, S. & Rogaway, P. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, 503–513 (Association for Computing Machinery, New York, NY, USA, 1990). URL `https://doi.org/10.1145/100216.100287`.

[23] Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* **31**, 469–472 (1985).

[24] Dong, C. & Loukides, G. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Transactions on Information Forensics and Security* **12**, 2792–2806 (2017).

[25] Kolesnikov, V., Rosulek, M., Trieu, N. & Wang, X. Scalable private set union from symmetric-key techniques. In *International Conference on the Theory and Application of Cryptology and Information Security*, 636–666 (Springer, 2019).

[26] Bar-Yossef, Z., Jayram, T., Kumar, R., Sivakumar, D. & Trevisan, L. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, 1–10 (Springer, 2002).

[27] Broder, A. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, 21–29 (1997).

[28] Flajolet, P., Fusy, É., Gandouet, O. & Meunier, F. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, 137–156 (Discrete Mathematics and Theoretical Computer Science, 2007).

[29] Kane, D. M., Nelson, J. & Woodruff, D. P. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 41–52 (2010).

[30] Heule, S., Nunkesser, M. & Hall, A. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, 683–692 (2013).

[31] Lindell, Y. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography* 277–346 (2017).