Experimental Analysis of Privacy-Preserving Statistics Computation*

Hiranmayee Subramaniam[1], Rebecca N. Wright[2], and Zhiqiang Yang[2]

[1] Stevens Institute of Technology graduate, hiran@polypaths.com.
[2] Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ, 07030, USA. rwright|zyang@cs.stevens-tech.edu.

Abstract. The recent investigation of privacy-preserving data mining and other kinds of privacy-preserving distributed computation has been motivated by the growing concern about the privacy of individuals when their data is stored, aggregated, and mined for information. Building on the study of *selective private function evaluation* and the efforts towards practical algorithms for privacy-preserving data mining solutions, we analyze and implement solutions to an important primitive, that of computing statistics of selected data in a remote database in a privacy-preserving manner. We examine solutions in different scenarios ranging from a high speed communications medium, such as a LAN or high-speed Internet connection, to a decelerated communications medium to account for worst-case communication delays such as might be provided in a wireless multihop setting.

Our experimental results show that in the absence of special-purpose hardware accelerators or practical optimizations, the computational complexity is the performance bottleneck of these solutions rather than the communication complexity. We also evaluate several practical optimizations to amortize the computation time and to improve the practical efficiency.

1 Introduction

Privacy-preserving data mining, as well as other kinds of privacy-preserving distributed computation, is intended to address conflicting goals. On the one hand, it is often desirable to extract information from collected data. On the other hand, there are often legitimate concerns about the privacy of personal data, proprietary data, and other sensitive information. Privacy-preserving data mining, in which certain computations are allowed, while other information is to remain protected, was first introduced in 2000 by Agrawal and Srikant [2] and Lindell and Pinkas [13]. Since then, extensive research has been devoted to

* This research was partially supported by the National Science Foundation (CCR-0331584), the Wireless Network Security Center (WiNSeC) at Stevens Institute of Technology, the New Jersey Commission on Science and Technology, and the NJ Center for Wireless Networking and Internet Security.

In *Proceedings of the Workshop on Secure Data Management*, held in conjunction with VLDB '04, August 30, 2004.

privacy-preserving data mining and other privacy-preserving computations efficient enough to be used on extremely large data sets (e.g., [3, 9, 5, 8, 17, 12, 7, 18, 10, 1, 19]).

In general, this research has been divided into solutions that provide strong cryptographic privacy protection, which require more computational overhead and have so far been limited to extremely simple (but useful) functions, and those that use perturbation, which provide weaker privacy properties, but allow much more efficient solutions and allow computation of more sophisticated data mining functions.

Our work provides an experimental evaluation of a cryptographic solution presented by the second author and others [5]. They introduced *selective private function evaluation*, a general methodology for efficient privacy-preserving solutions of computations by a client over data in a remote database. Their general solutions can provide efficiency improvements whenever the number of data elements involved in the computation is significantly fewer than the total number of data elements. As a particular instance, they consider a client/server environment in which the client and the server engage in a secure computation to evaluate a statistical function. Their solutions provide strong privacy guarantees, and involve encryption as a primary component.

As a specific selective private function computation, they consider private sum computation. In this setting, a client privately performs a sum or weighted sum of selected database elements held by the server. This is an important example because such protocols immediately yield private solutions for computing means, variances, and weighted averages, which can be useful on their own or as part of a larger privacy-preserving distributed data mining protocol. In our work, we implement a particular privacy-preserving solution to the private sum computation [6]; this protocol is described in more detail below. This protocol, as well as some of the others of Canetti et al. [5], can easily be extended to work for multiple distributed databases.

Our results show that the total running time needed is quite high, but it becomes feasible if certain straightforward optimizations are done, such as some client precomputation before the actual computation is to be done. Unless special hardware accelerators or practical optimizations are used, the computational delay caused by the encryption operations is the bottleneck, while the communication delay is significantly less.

To our knowledge, our implementation is one of the first implementations of privacy-preserving database computations. Relatedly, Malkhi et al.'s recent implementation [14] of Yao's general secure two-party computation solution [20] provides the first general secure multiparty computation results, and demonstrates that many computations on relatively small data sets can be done extremely efficiently. Indeed, secure multiparty computation and cryptographically strong privacy-preserving database computations, largely considered only theoretical, seem to be on the cusp of practicality as both theoretical and technological advances have improved their performance. Therefore, this kind of initial experimental work is an important contribution to understanding where such

results are within the realm of practice and where further improvements are still needed.

In Section 2, we describe the private selected sum problem and our implemented solution in more detail. We present our experimental results, including various practical optimizations that reduce the execution time, in Section 3.

2 Private Selected Sum Computation

We consider the simple problem of privately evaluating the sum of a subset of numbers. The server holds a database of n numbers. The client is interested in the sum of m selected numbers in the database (whose indices it is assumed to know, e.g., from some publicly available source), but the client does not wish to reveal its selection criteria. The database owner on the other hand wants to reveal to the client only the sum and not the individual elements that contribute to the sum.

A privacy-preserving client/server computation must satisfy three requirements [5]. *Correctness* states that as long as the client and server follow the protocol then the client's output is the correct value. *Client Privacy* requires that a malicious server cannot learn anything from the interaction about which values the client has selected to be involved in the computation. *Database Privacy* requires that the client learn only a predefined amount of information about the data.

A trivial but nonprivate solution to this problem is to let the client send the m indices in which it is interested to the database server. The server then computes the sum of the values at the specified indices and returns the sum to the client. While this solution preserves the privacy of the server, the server learns the set of indices the client is interested in, thus compromising the client privacy requirement. Conversely, another alternative would be for the server to expose the database to the client and have the client compute the sum of the numbers it is interested in. In this solution, the client's privacy is preserved but the client learns the entire contents of the server's database, and hence the goal of database privacy is not met.

Secure multiparty computation (SMC) is a powerful cryptographic primitive in which two or more parties can jointly compute a specified function of their input while hiding their inputs from one another. The problem of securely evaluating the selected sum is a specific example of SMC: the client and server wish to jointly evaluate the sum of a selected subset of numbers without the server revealing the individual elements or the client revealing the indices of interest. General SMC solutions [4, 11, 20] can provide solutions to the database sum problem providing both client and database privacy, but these solutions have communication overhead that is at least quadratic in the size of the database, which will generally be impractical for large databases. For example, initial results of the Fairplay system [14] suggest that straightforward implementation of Yao's solution would require an execution time of at least 15 minutes for a database of only 10,000 elements [16].

Canetti et al. [5] present cryptographic privacy-preserving solutions that in particular focus on reducing the communication. This focus is justified because strong privacy requires at least linear computation, as at a minimum every data element must be accessed in order to avoid leaking any information to the server. They present both linear-communication and sublinear-communication solutions.

As a starting point for our investigations of the practical performance of selective private function evaluation, we investigate a simple linear-communication solution that provides database privacy and client privacy using semantically secure homomorphic encryption [6]. Semantic security means that ciphertexts yield no information about their plaintexts. (In particular, encryption is randomized, and it is not possible to tell from two ciphertexts whether they encrypt the same plaintext or different plaintexts.) A homomorphic encryption scheme is an encryption scheme in which certain efficient computations on ciphertexts, which can be computed without knowledge of the plaintexts or the secret key, correspond to certain computations on plaintexts. For our protocol, we require a homomorphic encryption scheme satisfying: $E(a) \cdot E(b) = E(a+b)$, where $\cdot$ and $+$ denote modular multiplication and addition, respectively. It also follows that $E(a)^c = E(a \cdot c)$ for $c \in \mathbb{N}$. The Paillier cryptosystem [15] satisfies this property and is the cryptosystem of our choice in our implementation.

In the database sum setting, the server holds a database of n numbers $x_1, \ldots, x_n$. The client holds the set of indices $I_1, \ldots, I_n$, which represent the subset of numbers it is interested in. That is, I_i is 1 if x_i is to be included in the sum computation, and 0 otherwise.[1] (If desired, integer weights in some larger range could be used to produce a weighted sum, which in turn could be used for a weighted average.) The client has a public encryption key E and the corresponding private decryption key D of a homomorphic encryption scheme.

The private protocol, illustrated in Figure 1, executes as follows. The client encrypts its array of indices using the homomorphic cryptosystem and sends the encryptions $E(I_1), E(I_2), \ldots, E(I_n)$ to the server. The server then computes the product $\prod_{i=1}^{n} E(I_i)^{x_i}$. That is, the server takes the ith received encrypted value and raises it to the value of its ith data element x_i. Then the server multiplies all these values together modulo M, where M is a parameter of the encryption scheme. Note that this operation is applied directly to the received encrypted values, and does not require decryption nor does it yield any information about the cleartexts to the server. By the properties of homomorphic encryption, the resulting product is equal to the sum of numbers in the locations specified by the client's indices; that is,

$$\prod_{i=1}^{n} E(I_i)^{x_i} = E\left(\sum_{i=1}^{n} I_i x_i\right),$$

as desired. The server sends the product to the client, which decrypts it using the private key D to learn the desired sum. All operations are performed modulo M,

[1] The version of this paper published in the SDM proceedings mistakenly had this backwards. It is corrected here.

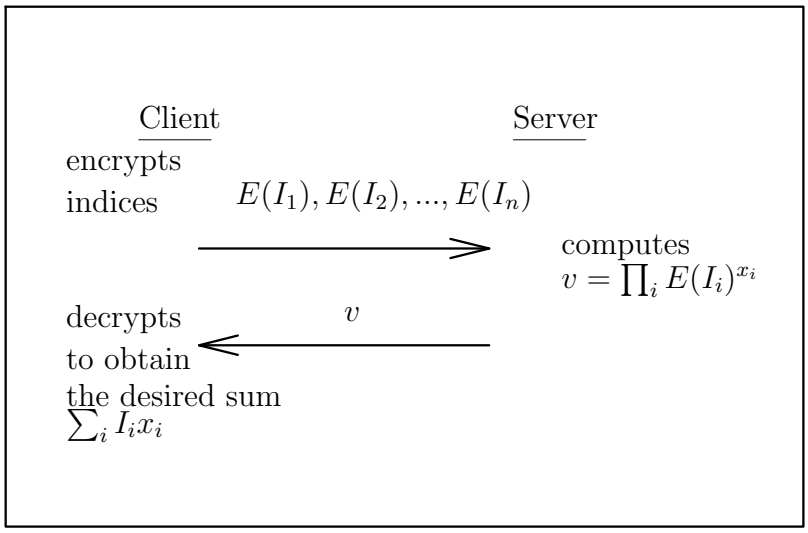

Fig. 1. Selected Sum Protocol

where M is a parameter of the homomorphic encryption cryptosystem used. The client's privacy is protected by the encryption of the indices, while the database's privacy is protected because the result sent back is the encryption of the desired sum, and does not contain any information about the other database values.

3 Experimental Results

We implemented the client/server protocol shown in Figure 1 and measured the computation and communication performance. We implemented the protocol in Java and C^{++}. The Java version uses the Java security package to perform cryptographic operations and the C^{++} implementation uses the OpenSSL libraries. Cryptographic keys are 512 bits. We experimented across various database sizes from 10,000 numbers to 100,000 numbers, with numbers of 32 bits each. On average, the performance results from our Java experiments were around five times slower than those of similar C^{++} experiments; except in Section 3.5, we report only the C^{++} numbers here.

The experimental data was measured on a High Performance Cluster at Stevens Institute of Technology in Hoboken, NJ and on a High Performance Cluster at Illinois Institute of Technology in Chicago, IL to measure communication complexity over short and long distances, respectively. Communication between the client and server was enabled by a 64Gbps switch within the High Performance Computing facility at Stevens; communication between the client in Chicago and the server in Hoboken used a 56Kbps modem. Our results show that despite the longer distance between the client and server and the decelerated

communication medium, computation time still prevails over the communication time, accounting for the bulk of the total running time.

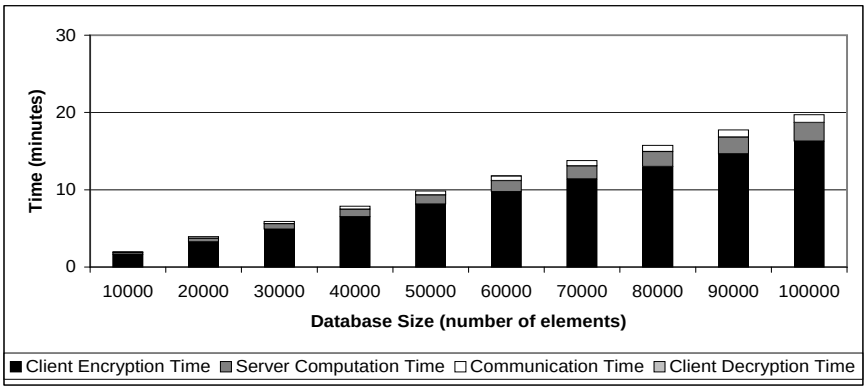

Fig. 2. Components of Overall Runtime without Any Optimizations over a Short Distance

3.1 Performance Results without Any Optimizations

Figures 2 and 3 show experimental results of the direct implementation of the solution described in Section 2, without any optimizations.

In Figure 2, both the client and the server processes ran on 2GHz Pentium-III processors with 3GB memory, connected by a high-performance gigabit network switch. Our results illustrate linear time performance, as expected. In this case, the bulk of the execution time is attributable to the client computation of the n public key encryptions of its index vector. The time for the server's computation is significantly less, followed by the communication time. The client's decryption time is constant (independent of the database size) and negligible since it is simply the time taken to decrypt a single encryption (of the desired sum). For a database of 100,000 elements, approximately 20 minutes is required for the execution.

Figure 3 shows the results of the experiment carried out over a long distance. In these experiments, the client process ran on a 500 MHz UltraSparc processor machine in Chicago, IL, and the server ran on a 1GHz Intel Pentium processor in Hoboken, NJ. Communication between client and server was via a 56Kbps dialup connection. As before, the client's encryption time increases linearly with increase in database size, as does the server's computation time and the communication time. As expected, the server's communication time now becomes a more substantial part of the execution time. However, despite the slow communication rate, the computation delay remains more significant than the communication delay.

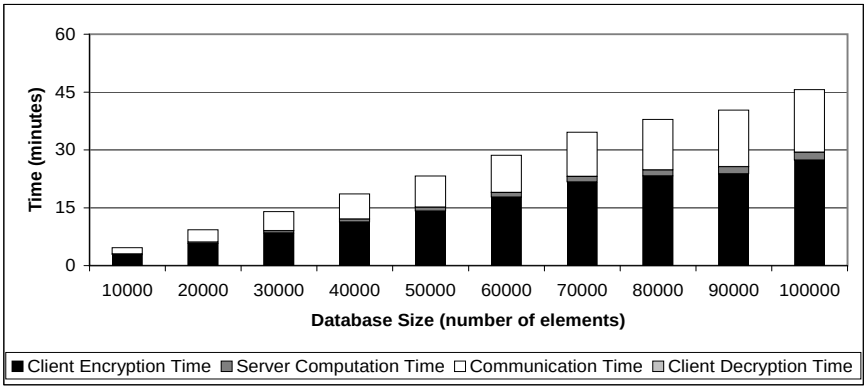

Fig. 3. Components of Overall Runtime without Any Optimizations Measured over a Long Distance

Our results show that in the absence of any practical optimizations or specialized hardware to accelerate client encryption, computation time is the bottleneck for the algorithm's performance. In Sections 3.2–3.5, we evaluate several straightforward practical optimizations.

3.2 Single-pass and Pipeline Parallelism

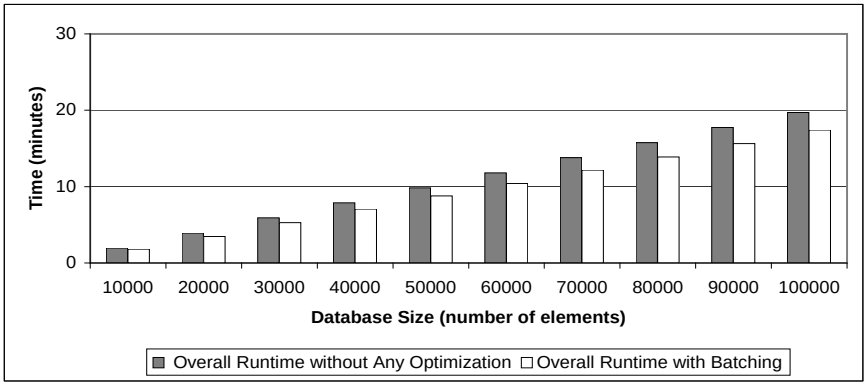

Fig. 4. Comparison of Overall Runtimes with and without Batching of Index Vector over a Short Distance

Noting that both the client computation and the server computation can be done in a single pass through their inputs, we implemented "batching" of the client processing, in which the client batches its processing of indices into smaller sized chunks, performing and sending the encryptions of the indices in

each chunk before proceeding to the next chunk. On receiving each chunk, the server can continue computing the partial product.

In addition to taking advantage of pipeline parallelism, this approach also reduces the memory requirements of both the client and server. At any point in time, the client has to allocate memory needed to hold only one chunk of its indices rather than the whole index vector. Similarly, the server need only hold a single database chunk in memory at one time. The optimal chunk size will depend on the relative communication and computation speeds, as well as the overhead in processing messages and memory access. In order to achieve maximum parallelization, ideally all three activities (communication of one batch, client processing of the next batch, and server processing of the previous batch) will require approximately the same amount of time.

Figure 4 compares the overall runtime of the protocol with and without batching of index vector. In our experiments, we took a batch size of 100 elements, resulting in approximately a 10% reduction in overall runtime.

3.3 Preprocessing the Index Vector

This optimization aims at reducing the computation complexity of the client by encrypting the indices offline in advance and storing the encrypted indices. Even if the client does not yet know which indices will be 0 and which will be 1, it can simply encrypt a large number of 0's and a large number of 1's to use later. When the client needs to send encrypted indices to the server, it can just retrieve the appropriate encryptions. The optimization is useful for mobile devices, e.g. PDAs, that have limited computing power but reasonable amounts of storage.

The results of this optimization are shown in Figure 5, with overall on-line execution times reduced to about $3\frac{1}{2}$ minutes for a database of 100,000 elements. The client's processing time, now simply to read the stored encryptions and send them to the server, is much smaller. All other components remain unchanged; the server's computation time becomes the dominant factor. This experiment was conducted on the high performance cluster with a 64Gbps bandwidth switch as the communications medium. Hence the delay in communication does not assume significant proportions. The reduction in overall runtime is about 82%.

Figure 6 shows the results observed over a 56Kbps dialup connection with the client at Chicago, IL and the server at Hoboken, NJ. In this case, the communication delay becomes the significant factor.

3.4 Combination of Optimizations

The batching of index vector optimization reduces the server's idle time while preprocessing the vector of indices reduces the client's on-line encryption time. Combining these optimizations results in an overall on-line runtime reduction of about 94%, as shown in Figure 7.

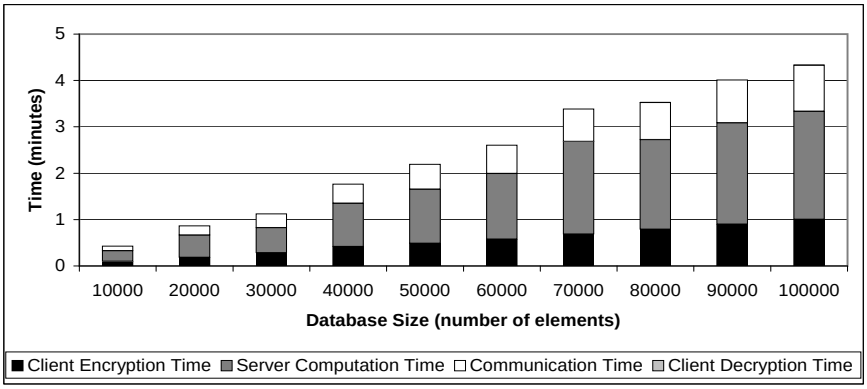

Fig. 5. Components of Overall Runtime after Preprocessing the Index Vector over a Short Distance

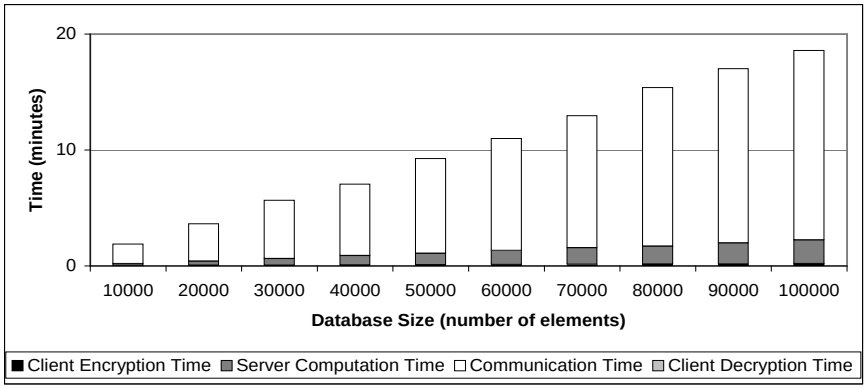

Fig. 6. Components of Overall Runtime after Preprocessing the Index Vector Measured over a Long Distance

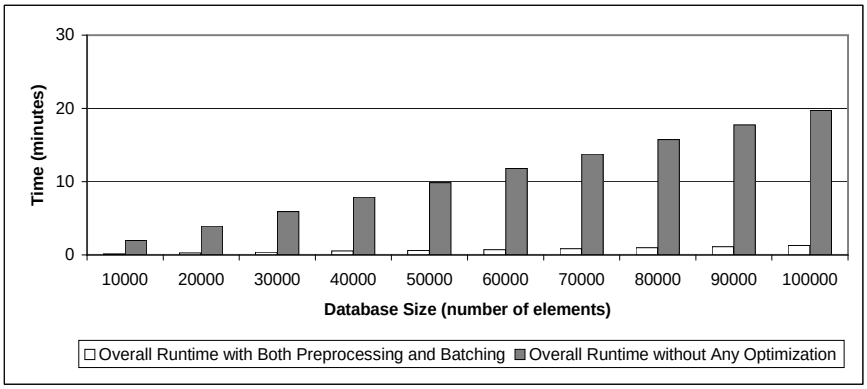

Fig. 7. Performance Gain Due to Combination of Optimizations over a Short Distance

3.5 Using Multiple Clients in Parallel

This alternative aims at reducing the time spent by the client in encrypting the index vector by partitioning the task of encryption among multiple clients. The challenge is how to protect the privacy of the server while using multiple clients.

In this setting, k clients work in cooperation. Each client is responsible for $1/k$th of the database, and will interact with the server to learn a partial sum corresponding to the chosen indices in that part of the database. However, learning these partial sums violates database privacy. Accordingly, the server uses a randomized blinding to protect the partial sums; the blinding is removed by the clients only after the partial sums are combined into a single sum, as shown in Figure 8 for $k = 3$.

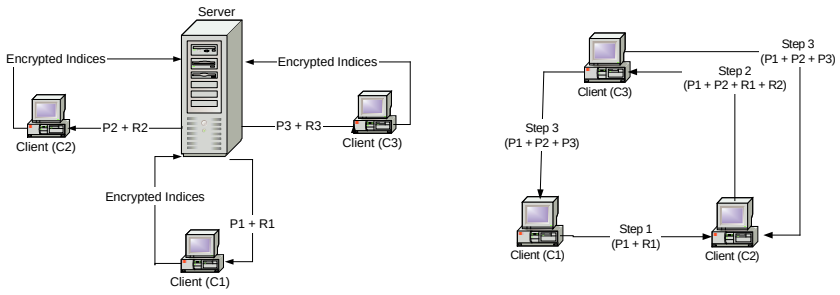

Fig. 8. Multiple Clients ($k = 3$)

In phase one, k clients C_1, C_2, ..., C_k are involved each holding an index vector of size n/k elements. (We assume for simplicity that the database size n is a multiple of k.) The clients independently and in parallel choose their own encryption keys and interact with the server to learn a blinded encryption of the appropriate partial sum. That is, the server chooses random numbers R_1, R_2, .., R_k such that $\sum_{i=1}^{k} R_i = 0 \pmod{M}$ (where again M is a parameter of the encryption scheme). When computing the product to return to client C_i, the server also computes $E(R_i)$ and multiplies it into the product. This has the effect of adding R_i to the partial sum P_i.

In phase two, the clients combine their partial sums and remove the blinding factor:

1. Client C_1 sends its blinded partial sum to client C_2.
2. In turn, each client C_i adds the value received from client C_{i-1} to its own blinded sum and sends the result to client C_{i+1}.
3. Client C_k receives the blinded partial sum from client C_{k-1}, adds it to its blinded partial sum to generate the total unblinded sum, and broadcasts the result to all the other clients.

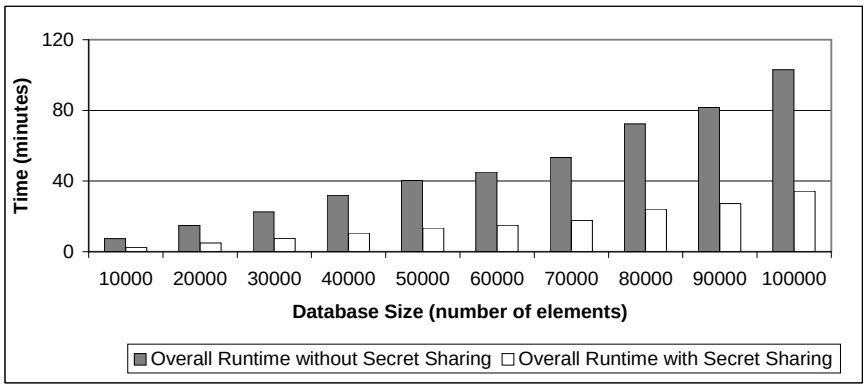

Fig. 9. Performance Improvement Due to Secret Sharing with Three Clients (Java implementation)

The results in Figure 9 show performance results for $k = 3$. The overall execution time is reduced by a factor of approximately 2.99, which represents a 3-fold improvement, minus a small overhead for the combining phase. Note that we implemented multiple clients only for our Java implementation, so these performance numbers are significantly higher than those in earlier graphs. They are shown only to indicate the close to 3-fold improvement. The use of k clients would result in approximately a k-fold reduction in execution time.

4 Conclusions

We have analyzed and implemented an instance of selective private function evaluation that privately computes the sum of a subset of numbers held by a remote database, where the selection of the subset is done by the client. The database does not learn anything about which values the client's computation involves, and the client does not learn anything about the values in the database other than what is implied by the value of the given sum.

Our experimental results show that the running time needed is quite high, though perhaps feasible in some settings where privacy is considered sufficiently important. In a direct implementation, overall running times are around 20 minutes for a database of 100,000 elements in a high-speed communication environment. With straightforward optimizations, the running times are only a few minutes, and may be within the realm of practice. Unless practical optimizations or specialized hardware are used to accelerate encryptions, computation delay is the major bottleneck of performance of our implementation.

It remains open to improve the execution times to scale efficiently to realistically-sized databases. As directions for future work, we plan to investigate the use of special-purpose cryptographic hardware, as well as methods that give up some quantifiable amount privacy in order to achieve significant performance improvements.

Acknowledgments

This research was supported in part by the National Science Foundation (CCR-0331584), the Wireless Network Security Center (WiNSeC) at Stevens Institute of Technology, the New Jersey Commission on Science and Technology, and the NJ Center for Wireless Networking and Internet Security. Additional resources were provided by the Stevens High Performance Computing Facility. We also thank Scalable Computing Software laboratory at Illinois Institute of Technology, Chicago, IL for granting us access to their computing facility.

References

1. G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k^{th}-ranked element. In *Proc. Eurocrypt 2004*, LNCS 3027, pages 40–55. Springer-Verlag, 2004.
2. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press, May 2000.
3. M. Atallah and W. Du. Secure multi-party computational geometry. In *Proc. 7th International Workshop on Algorithms and Data Structures*, pages 165–179. Springer-Verlag, 2001.
4. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10. ACM Press, 1988.
5. R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proc. 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 293–304. ACM Press, 2001.
6. R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Personal communication, 2003.
7. A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proc. 22nd Symposium on Principles of Database Systems*, pages 211–222. ACM Press, 2003.
8. A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228. ACM Press, 2002.
9. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. In *Proc. 28th International Colloquium on Automata, Languages and Programming*, pages 927–938. Springer-Verlag, 2001.
10. M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proc. Eurocrypt 2004*, LNCS 3027, pages 1–19. Springer-Verlag, 2004.
11. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. 19th Annual ACM Conference on Theory of Computing*, pages 218–229. ACM Press, 1987.
12. M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, June 2002.
13. Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002. An earlier version appeared in *Proc. Crypto 2000*.

14. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. Usenix Security Symposium 2004*, 2004. To appear.
15. P. Paillier. Public-key cryptosystems based on composite degree residue classes. *In Advances in Cryptography - EUROCRYPT 99*, pages 223–238, 1999.
16. Y. Sella. Personal communication, 2004.
17. J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644. ACM Press, 2002.
18. J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215. ACM Press, 2003.
19. R. N. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2004. To appear.
20. A. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.