SECURE OUTSOURCING OF SCIENTIFIC COMPUTATIONS

Mikhail J. Atallah and John R. Rice
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907, U.S.A.
email: {mja, jrr}@cs.purdue.edu

December 7, 1998

Abstract

We investigate the *outsourcing* of numerical and scientific computations using the following framework: A *customer* who needs computations done but lacks the computational resources (computing power, appropriate software, or programming expertise) to do these locally, would like to use an external *agent* to perform these computations. This currently arises in many practical situations, including the financial services and petroleum services industries. The outsourcing is *secure* if it is done without revealing to the external agent either the actual data or the actual answer to the computations. The general idea is for the customer to do some carefully designed local preprocessing (*disguising*) of the problem and/or data before sending it to the agent, and also some local postprocessing of the answer returned to extract the true answer. The disguise process should be as lightweight as possible, e.g., take time proportional to the size of the input and answer. The disguise preprocessing that the customer performs locally to "hide" the real computation can change the numerical properties of the computation so that numerical stability must be considered as well as security and computational performance. We present a framework for disguising scientific computations and discuss their costs, numerical properties, and levels of security. We show that no single disguise technique is suitable for a broad range of scientific computations but there is an array of disguises techniques available so that almost any scientific computation can be disguised at a reasonable cost and with very high levels of security. These disguise techniques can be embedded in a very high level, easy-to-use system (problem solving environment) that hides their complexity.

Table of Contents

1. INTRODUCTION

1. Outsourcing and Disguise

Outsourcing is a general procedure employed in the business world when one entity, the *customer*, chooses to farm out (*outsource*) a certain task to an external entity, the *agent*. The reasons for the customer to outsource the task to the agent can be many, ranging from a lack of resources to perform the task locally to a deliberate choice made for financial or response time reasons. Here we consider the outsourcing of numerical and scientific computations, with the added twist that the problem data and the answers are to be hidden from the agent who is performing the computations on the customer's behalf. That is, it is either the customer who does not wish to trust the agent with preserving the secrecy of that information, or it is the agent who insists on the secrecy so as to protect itself from liability because of accidental or malicious (e.g., by a bad employee) disclosure of the confidential information.

The current outsourcing practice is to operate "in the clear", that is, by revealing both data and results to the agent performing the computation. One industry where this happens is the financial services industry, where the proprietary data includes the customer's projections of the likely future evolution of certain commodity prices, interest and inflation rates, economic statistics, portfolio holdings, etc. Another industry is the energy services industry, where the proprietary data is mostly seismic, and can be used to estimate the likelihood of finding oil or gas if one were to drill in a particular geographic area. The seismic data is so massive that doing matrix computations on such large data arrays is beyond the computational resources of even the major oil service companies, which routinely outsource these computations to supercomputing centers.

We consider many science and engineering computational problems and investigate various schemes for outsourcing to an outside agent a suitably disguised version of the computation in a way that hides the customer's information from the agent and yet the answers returned by the agent can be used to obtain easily the true answer. The local computations should be as minimal as possible and the disguise should not degrade the numerical stability of the computation.

2. The Difference Between Disguise and Encryption

The techniques presented here differ from what is found in the cryptography literature concerning this kind of problem. Secure outsourcing in the sense of [**?**] follows an information-theoretic approach, leading to elegant negative results about the impossibility of securely outsourcing computationally intractable problems. In contrast, our methods are geared towards scientific computations that may be solvable in polynomial time, (e.g., solution of a linear system of equations) or where time complexity is undefined (e.g., the work to solve a partial differential equation is not related to the size of the text strings that define the problem). In addition, the cryptographic protocols literature contains much that is reminiscent of the framework of the present proposal, with many elegant protocols for cooperatively computing functions without revealing information about the functions' arguments to the other party (cf. the many references in, for example, [**?**, **?**]). The framework of the *privacy homomorphism* approach that has been proposed in the past [**?**] assumes that the outsourcing agent is used as a permanent repository of the data, performing certain operations on it and maintaining certain predicates, whereas the customer needs only to decrypt the data from the external agent's repository to obtain from it the real data. Our framework is different in the following ways:

- The customer is not interested in keeping data permanently with the outsourcing agent; instead, the customer only wants to use temporarily its superior computational resources.
- The customer has some local computing power that is not limited to encryption and decryption. However, the customer does not wish to do the computation locally, perhaps because of the lack of computing power or appropriate software or perhaps because of economics.

Our problem is also reminiscent of the *server-aided computation* work in cryptography, but there most papers deal with modular exponentiations and not with numerical computing [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**].

Our problems and techniques afford us (as will be apparent below) the flexibility of using *one-time-pad* kinds of schemes for disguise. For example, when we disguise a number x by adding to it a random value r, then we do not re-use that same r to disguise another number y (we generate another random number for that purpose). If we hide a vector of such x's by adding to each a randomly generated r, then we have to be careful to use a suitable distribution for the r's. Throughout this paper when we use random numbers, random matrices, random permutations, random functions (e.g., polynomials, splines, etc., with random coefficients) etc.; it is always assumed that each is generated independently of the others, and that quality random number generation is used (cf. [**?**, Chap. 23], [**?**, Chap. 12], [**?**, **?**, **?**]).

The parameters, types, and seeds of these generators provide the keys to the disguises. We show how to use a single key to generate multiple keys which are "independent" and which simplify the mechanics of the disguise techniques. This key is analogous to the key in encryption but the techniques are different.

The following simple example illustrates the difference between encryption and disguise. Consider a string F of text characters that are each represented by an integer from 1 to 256 (i.e., these are byte strings). Suppose that F_1 is an encryption of F with one of the usual algorithms. Suppose that F_2 is a disguise of F that is created as follows: (1) Choose a seed (the disguise key) for a uniform random number generator and create a sequence G of random integers between 0 and 128, (2) Set $F_2 = F_1 + G$. Assume now that F is a constant (the single value 69) string of length N and the agent wishes to discover the value of this constant. It is not possible to discover from F_1 the value 69 no matter how large N is. However, it is possible to discover 69 from F_2 if N is large enough. Since G is uniform, the mean of the values of G converge to 64 as N increases and thus, as N increases, the mean of F_2 converges to $133 = 64 + 69$ and the rate of convergence is order $1/\sqrt{N}$. Thus, when $1/\sqrt{N}$ is somewhat less than $1/2$, we know that the mean of F_2 is 133 and that the character is 69. An estimate of N is obtained by requiring that $128/\sqrt{N}$ be less than $1/2$ or N be more than about 60–70,000.

The point of this example is that the encription cannot be broken in this case without knowing the encryption key — even if one knows the encryption method. However, the disguise can be broken without knowing the key provided the disguise method is known. Of course, it follows that one should not use such a simplistic disguise and we provide disguise techniques for scientific computations with security comparable to that of the most secure encryptions.

3. Four Simple Examples

The nature and breadth of the disguises possible are illustrated by the following.

1. Matrix Multiplication

Consider the computation of the product of two $n \times n$ matrices M_1 and M_2. We use $\delta_{x,y}$ to denote the Kronecker delta function that equals 1 if $x = 1$ and 0 if $x \neq y$. The disguise requires six steps:

1. Create (i) three random permutations π_1, π_2, and π_3 of the integers $\{1, 2, \ldots, n\}$, and (ii) three sets of non-zero random numbers $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, $\{\beta_1, \beta_2, \ldots, \beta_n\}$, and $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$.
2. Create matrices P_1, P_2, and P_3 where $P_1(i,j) = \alpha_1 \gamma_{\pi_1(i),j}$, $P_2(i,j) = \beta_i \gamma \pi_2(i), j$, and $P_3(i,j) = \gamma_k \delta_{\pi_3(i),j}$. These matrices are readily invertible, e.g., $P_1^{-1}(i, ju) = (\alpha_j)^{-1} \gamma_{\pi_1^{-1}(i),j}$.
3. Compute the matrix $X = P_1 M_1 P_2^{-1}$. We have $X(i,u) = (\alpha_i / \beta_j) M_2(\pi_1(i), \pi_2(j))$.
4. Compute $Y = P_2 M_2 P_3^{-1}$.
5. Send X and Y to the agent which computes the product $Z = XY = (P_1 M_1 P_2^{-1})(P_2 M_2 P_3^{-1}) = P_1 M_1 M_2 P_3^{-1}$ and sends Z back to the customer.
6. Compute locally, in $O(n^2)$ time, the matrix $P_1^{-1} Z P_3$, which equals $M_1 M_2$.

This disguise may be secure enough for many applications, as the agent would have to guess two permutations (from the $(n!)2$ possible such choices) and $3n$ numbers (the α_1, β_i, γ_i) before it can determine M_1 or M_2. This example is taken from [**?**] where a much more secure disguise is presented along with disguises for other linear algebra procedures (convolutions, solution of linear systems, FFTs). All of these disguises require $O(n^2)$ computation which is the minimum possible since the problem involve $O(n^2)$ data. The outsourced computations require $O(n^3)$ operations.

2. Quadrature

The objective is to estimate

$$\inf_{1}^{b} f(x) dx$$

with accuracy *eps*. The disguise is as follows.

1. Choose $X_a = a, X_7 = b$ and 5 ordered, random numbers X_i in $[a, b]$ and 7 values v_i with $\min |f(x)| \approx M_1 \leq M_2 \approx \max |f(x_i)|$. ($M_1$ and M_2 are only estimate roughly).
2. Create the cubic spline $g(x)$ with knots X_i so that $g(X_i) = V_i$.

3. Integrate $g(x)$ exactly from a to b to obrtain I.

4. Send $g(x) + f(x)$ and eps to the agent for numerical quadrature and recive the value I_2 back.

5. Compute $I_2 - I_1$ which is the answer.

All the computations made locally are simple, of fixed work, and independent of $f(x)$ and eps. The random vectors and matarices of the previous example are replaced by a “random” smooth function. ONe has to determine 12 random numbers in order to break the disguise.

3. Edge Detection

The objective is to determine the edges in picture represented by an $n \times n$ array of pixel values $p(x, y)$ between 0 and 100,000 on the square $0 \leq x_1 y \leq 1$. This disguise is as follows:

1. Set $x, y = 0$, $X_{10}, Y_{10} = 1$, choose two sets of 8 ordered, random numbers with $0 < V_i, Y_i < 1$, choose 100 random values $0\ leqV_{i,u} \leq 50,000$, and choose four pairs (a_i, b_i) of positive, random numbers with $a_1 = \min a_i$, $a_4 = \max a_i$, $b_1 = \min b_i$, $b_4 = \max b_i$.

2. Create the bi-cubic spline $s(x, y)$ so that $s(x_i, y_j) = V_{ij}$.

3. Determine the linear change of coordinates from (X, Y) to (U, V) that maps the unit square into the quaddrilateral with vertices (a_i, b_i).

4. Send $p(u(x, y), v(x, y)) + s(u(x, y), v(x, y))$ to the agent for edge detection and receive the image $e(u, v)$ back showing the edges.

5. Compute $e_1(x(u, v), y(u, v))$ to obtain the edges.

This disguise uses the fact that adding a smooth pixel function to the image and making a smooth change of coordinates does not add or delete any edges from the image. As in 1.3.1, the work of the disguise is proportional to the size of the data for the computing (plus a small constant amount). Here one must determine 124 random numbers in order to break the disguise.

4. Solution of a Differential Equation

The objective is to solve the two point boundary value problem

$$y'' + a_1(x)y' + a_2(x)Y = f(x, y)y(s) = Y_0, Y(b) = Y_i$$

The disguise is as follows.

1. Choose a spline $g(x)$ as in 1.3.2 above.

2. Create the function
$$u(x) = g'' + a_1(x)g' + a_2(x)g$$

3. Send the problem

$$y'' + a_1(x)Y' + a_2(x)g = f(x,y) + u(x)y(a) = 1/0 + u(a), y(b) = y_1 + u(b)$$

   to the agent for solution and recieve $z(x)$ back

4. Compute $z(x) - g(x)$ which is the solution.

This disguise applies the problem's mathematical oerator to a known function and then combines the real and the artificial problems. Here one must determine 12 random numbers to break the disguise.

This paper is organized as follows. In Section 2 the general framework for disguise of scientific problems is presented including atomic disguise techniques, key management and "programming" disguises. The detailed specifications of these disguises are given in Appendix A. Section 3 presents applications of these techniques to five broad areas of scientific computation. The details for these applications are given in Appendix B. The security of disguises is analyzed in Section 4 for attacks using statistical, approximation theoretic and symbolic code analysis methods.

References

[1] M. Abadi, J. Feigenbaum, J. Killian, On hiding information from an oracle, *J. of Computer and System Sciences*, **39**, pages 21–50, 1989.

[2] M.J. Atallah, K.N. Pantazopoulos, E.H. Spafford. Secure outsourcing of some computations, Department of Computer Sciences CSD-TR-96-074, Purdue University, 1996.

[3] P. Beguin, J-J. Quisquater. Fast server-aided RSA signatures secure against active attacks, *CRYPTO*, (1995), 57–69.

[4] B. Dole, S. Lodin, and E. H. Spafford. Misplaced trust: Kerberos 4 session keys, in *Proceedings of 4th Symposium on Network and Distributed System Security*, IEEE Press, (1997), 60–71.

[5] D. E. Eastlake, S. D. Crocker, and J. I. Schiller, *RFC–1750 Randomness Recommendations for Security*, Network Working Group, December (1994).

[6] S. Garfinkel and E. H. Spafford. *Practical UNIX & Internet Security*, O'Reilly & Associates, second edition, (1996).

[7] S-J. Hwang, C-C. Chang, W-P. Yang. Some active attacks on fast server-aided secret computation protocols for modular exponentiation, *Cryptography: Policy and Algorithms*, LNCS 1029, (1996), 215–228.

[8] S-I. Kawamura, A. Shimbo. Fast server-aided secret computation protocols for modular exponentiation, *Proc. of IEEE J. on Selected Areas in Communications*, **11**(5), (1993), 778–784.

[9] D. E. Knuth, *The Art of Computer Programming, Volume 2*, Addison Wesley, second edition, 1981.

[10] C-S. Laih, S-M. Yen, Secure addition sequence and its application on the server-aided secret computation protocols, *AUSCRYPT*, (1992), 219–230.

[11] C-H. Lim, P. J. Lee. Security and Performance of server-aided RSA computation protocols, *CRYPTO*, (1995), 70–83.

[12] T. Matsumoto, K. Kato, H. Imai, Speeding up secret computations with insecure auxiliary devices, *CRYPTO*, (1988), 497–506.

[13] B. Pfitzmann, M. Waidner, Attacks on protocols for server-aided RSA computation, *EUROCRYPT*, (1992), 153–162.

[14] J-J. Quisquater, M. de Soete, Speeding up smart card RSA computations with insecure coprocessors, *Smart Card 2000*, North Holland, (1991), 191–197.

[15] R. L. Rivest, L. Adleman, and M. L. Dertouzos, On data banks and privacy homomorphisms, in (Richard A. DeMillo, editor), *Foundations of Secure Computation*, Academic Press, (1978), 169–177.

[16] B. Schneier, *Applied Cryptography*, Wiley, second edition, (1996).

[17] A. Shimbo, S. Kawamura, Factorization attacks on certain server-aided computation protocols for the RSA secret transformation, *Electronic Letters*, **26**(17), (1990), 1387–1388.

[18] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, Boca Raton, FL, (1995).

[19] G. J. Simmons, editor, *Contemporary Cryptology: The science of Information Integrity*, IEEE Press, (1992).