OUTSOURCING SCIENTIFIC COMPUTATIONS SECURELY*

Mikhail J. Atallah and John R. Rice
CERIAS: Center for Education and Research in Information Assurance and Security
Purdue University, West Lafayette, IN 47907, U.S.A.
email: {mja, jrr}@cs.purdue.edu

Abstract

The *outsourcing* of numerical and scientific computations, as introduced in (Atallah *et al.*, 2001) uses the following framework: A *customer* needs computations done but lacks the computational resources (computing power, appropriate software, or programming expertise) to do these locally. An external *agent* can do these computations. 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 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 (*unveiling*) of the answer returned to extract the true answer. In this paper we extend this concept to the case of more than one customer, introducing the notion of *mutually secure outsourcing* where two or more parties contribute their private data into the (disguised) common computation performed through the external agent; the customers are to know the result but not each other's private data, and the external agent should know neither the private data nor the result. We review the framework for disguising scientific computations and discuss their applicability, costs, and levels of security. We also introduce techniques for the disguise of programs in general, not just those for scientific computations.

1 INTRODUCTION

In (Atallah *et al.*, 2001) we consider many science and engineering computational problems and present disguise schemes for outsourcing the computation in such a way that the customer's information is hidden from the external agent, and yet the answers returned by the agent can be used to obtain easily the true answer. The local computations are minimal – i.e., proportional to the size of the local input (which is unavoidable). The bulk of the computational task falls on the external agent (which would typically be an entity with considerable computing power, such as a supercomputing center). We note that even computations that depend on the exact relationships among the data items can be disguised for outsourcing. Examples include ordering a list of numbers and looking for a template in an image. In this paper we consider the case when the input data is spread among many customers who should learn nothing about each other's data (other than what they can infer from the computation's overal answer).

This framework is similar to, yet differs in significant ways from the traditional secure multi-party computation protocols (for a good introduction to these see Schneier, 1996, and its extensive bibliography). Some of these differences were mentioned in (Atallah *et al*, 2001), which also contains a review of related work in cryptography. Some significant differences are:

- Here we would like any superlinear computational burden to fall on the outsourcing (external) agent. The computations performed locally by the customers on their private data should be proportional to the size of their local data. The customers do not keep data permanently with the outsourcing agent; instead, a customer only uses temporarily the agent's superior computational resources.
- Here we would like the computational burden on the external agent to be $O(T)$, where T is the time it would take that agent to solve the problem *if it already had all of the data needed for the computation* (instead of the data being scattered among the customers).
- Whereas in cryptographic multi-party protocols every participant knows that "this is step i of run j of prococol X," here the external agent is told neither i nor j nor X; in fact an effort to obfuscate these is preferred, e.g., by interleaving the various steps/runs of multiple computations and/or by using more than one external agent for the various steps of a single computation.

*Portions of this work were supported by Grants DCR-9202807 and EIA-9903545 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

The security of a scheme should not depend on an assumption that the external agent is ignorant of the source code used by the customers for creating a disguise. Rather, the security should depend on the fact that the external agent does not know the customer's seeds (of the generators for the randomness).

The paper (Atallah *et al.*, 2001) gives schemes for secure outsourcing (just 2-party — a single customer) of the following computations. Those marked with an asterisk can be done in a mutually secure manner: Quadrature, solving a linear system*, matrix inversion, matrix multiplication*, computing convolutions*, sorting, template matching in image analysis (using the sum of pixelwise $(x-y)^2$ or $|x-y|$ as measures of closeness)*, and Matching string patterns*. The mutually secure versions of the above asterisk-marked problems are not all simple extensions of (Atallah *et al.*, 2001), but for page-limit reasons we do not include them all in this paper. Here we use the problem of solving a linear system to illustrate the ideas involved (the other asterisk-marked problems will be reported elsewhere). We also sketch here a solution for fingerprint matching, review the framework for disguising scientific computations and discuss their applicability/costs/security, and also introduce techniques for the disguise of programs in general (not just those for scientific computations).

Our problems allow the flexibility of using one-time-pad-like 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 r. If we hide a vector $\mathbf{x}$ by adding to it a random vector $\mathbf{r}$, then we have to be careful to use a suitable distribution for $\mathbf{r}$. The random numbers used for disguises are not stored locally and used to "undo" the effect of the disguise on the result from the agent. Randomness is also used to modify the nature of the disguise algorithm itself.

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 assumed that quality random number generation is used. The parameters, types, and seeds of these generators provide the keys to the disguises. A single key is occasionally used to generate multiple keys which are used "independently" and which simplify the mechanics of the disguise techniques. This key is analogous to the key in encryption but the techniques are different.

2 MUTUALLY SECURE OUTSOURCING

Mutually secure outsourcing allows several different customers to contribute problem information to a computation without revealing to anyone their part of the information. The bulk of the computational task should be carried out by yet another party, who is to remain ignorant of the customers' private data and of the computed solution. The technique is described in terms of two parties, A and B, who have information, $d1$ and $d2$, respectively, and who want another party, C, to make a computation to compute a result z. The technique can easily be extended to more parties. The goal is that A and B do not reveal the information $d1$ and $d2$ to anyone and C does not even learn the result of z. The procedure consists of one or more of the following basic rounds.

Basic Round of Mutually Secure Outsourcing

1. *The type and basic parameters of the computation are determined.*
2. *A mutually acceptable disguise program is created for this computation by A, B, or C.*
3. *A and B agree on a Master Key for this program.*
4. *A and B independently apply this program to d1 and d2 to produce disguised information $\hat{d}1$ and $\hat{d}2$ plus unveiling data v_1 and v_2.*
5. *A and B exchange the unveiling data in a secure fashion.*
6. *$\hat{d}1$ and $\hat{d}2$ are sent to C who computes $\bar{z}$ and returns $\bar{z}$ to A and B.*
7. *A and B apply the unveiling data v_1 and v_2 and reveal z.*

As in ordinary outsourcing, one prefers a reasonably small effort to disguise $d1$ and $d2$ and to unveil $\bar{z}$.

The nature and breadth of the disguises possible are illustrated by the mutually secure outsourcing of the solution of a linear system of equations. Assume that A and B each have part of the information (equations and right side). Thus the problem is to solve

$$\begin{pmatrix} M1 \\ M2 \end{pmatrix} z = \begin{pmatrix} b1 \\ b2 \end{pmatrix}$$

where A has $M1$ and $b1$ and B has $M2$ and $b2$. The lengths of the vectors $b1$ and $b2$ are N_1 and N_2, respectively. The disguise procedure in (Atallah *et al*, 2001) is modified by partitioning as follows:

1. Choose Master Key.

2. (a) A creates random permutation vectors $\pi 11$, $\pi 21$ and $\pi 31$ of length N_1 each. A also creates random vectors $\alpha 1$, $\beta 1$ and $\gamma 1$ of length N_1 each. A then uses these to create the $N_1 \times N_1$ weighted permutation matrices

$$P_{11}(i,j) = \alpha 1(i) \text{ if } \pi 11(i) = j; = 0 \text{ otherwise}$$

$$P_{21}(i,j) = \beta 1(i) \text{ if } \pi 21(i) = j; = 0 \text{ otherwise}$$

$$P_{31}(i,j) = \gamma 1(i) \text{ if } \pi 31(i) = j; = 0 \text{ otherwise}$$

(b) B creates random permutation vectors $\pi 12$, $\pi 22$ and $\pi 32$ of length N_2 each. B also creates random vectors $\alpha 2$, $\beta 2$ and $\gamma 2$ of length N_2 each. B then uses these to create the $N_2 \times N_2$ weighted permutation matrices

$$P_{12}(i,j) = \alpha 2(i) \text{ if } \pi 12(i) = j; = 0 \text{ otherwise}$$

$$P_{22}(i,j) = \beta 2(i) \text{ if } \pi 22(i) = j; = 0 \text{ otherwise}$$

$$P_{32}(i,j) = \gamma 2(i) \text{ if } \pi 32(i) = j; = 0 \text{ otherwise}$$

3. A and B agree on an index k and size K.

4. (a) A creates a random matrix of order $N1$ by K and then creates the matrix $B1$ by substituting $b1$ as the kth column of this matrix.

(b) B creates a random matrix of order $N2$ by K and creates the matrix $B2$ by substituting $b2$ as the kth column of this matrix.

5. (a) A computes $\hat{M}1 = P_{11}M1P_{21}^{-1}$, and $\hat{B}1 = P_{11}B1P_{31}^{-1}$.

(b) B computes $\hat{M}2 = P_{12}M2P_{22}^{-1}$, and $\hat{B}2 = P_{12}B2P_{32}^{-1}$.

6. The linear system of equations

$$\begin{pmatrix} \hat{M}1 \\ \hat{M}2 \end{pmatrix} \begin{pmatrix} \hat{z}1 \\ \hat{z}2 \end{pmatrix} = \begin{pmatrix} \hat{B}1 \\ \hat{B}2 \end{pmatrix}$$

is outsourced to C for solution and $(\hat{z}1, \hat{z}2)$ is returned by C.

7. (a) A computes $P_{21}^{-1}\hat{z}1P_{31}$ to obtain $z1$.

(b) B computes $P_{22}^{-1}\hat{z}2P_{32}$ to obtain $z2$.

(c) If desired, A and B may exchange P_{21}, P_{31} and P_{22}, P_{32} so that each may know the entire solution $(z1, z2)$.

Note that the above requires $O(n^2)$ time in local computations by the customers, which is the minimum possible since the problem involves $O(n^2)$ data. The outsourced computations, on the other hand, require $O(n^3)$ operations and their burden is borne by the outsourcing agent C.

The unveiling data used in the last step are the matrices P_{21}, P_{22}, P_{31} and P_{32}. Note that if A and B do not need to know the entire solution $(z1, z2)$ then they do not have to reveal any of the unveiling data to each other. Note, however, that if A and B do decide to share P_{21}, P_{22}, P_{31} and P_{32}, then the security of the disguise is weakened and one may have to use more complex (possibly dense) matrices P_{11} and P_{12} to give more strength to the disguise. If P_{11} and P_{12} are dense random matrices (for better security) then the matrix multiplications that used to be done locally by A and B (for disguising), and that took $O(n^2)$ local computation time, can no longer be done locally because they would now take $O(n^3)$ time. We can achieve $O(n^2)$ local computation time by A and B *even if* P_{11} *and* P_{21} *are dense* as follows. We illustrate this for A; a similar discussion holds for B. Whenever A needs to multiply locally two dense matrices X and Y, A instead engages, with an outside agent C', in the secure (just 2-party – not involving B) outsourcing method of [1] for computing XY. Recall from [1] that this is done without revealing to C' either X, Y, or XY, and that C' does the $O(n^3)$ time computations while A does only $O(n^2)$ work. It is desirable (but not required) to choose C' to be different from C.

3 DISGUISE TECHNOLOGIES

We note that multiple disguise techniques are necessary and identify five broad classes of disguises. Within each class there may be several or many *atomic disguises*, the technology used to create complete disguises. No single disguise technique is sufficient for the broad range of computations. The analogy with ordinary disguises is appropriate: one does completely different things to disguise an airplane hangar than one does to disguise a person. In a personal disguise one changes their hair, the face, the clothes, etc., using several different techniques. The same is true for scientific computation. The necessity for multiple disguises illustrates the different nature of disguise and encryption. Multiple disguises require multiple keys and thus we present a technique to use one *master key* from which many sub-keys may be generated automatically with the property that the discovery of one sub-key does not compromise the master key or any other sub-key.

Atomic Disguises. A disguise has three important properties: *Invertibility*: After the disguise is applied and the outsourced computation made, one must be able to recover the result of the original computation. *Security*: No one without the key of the disguise should be able to discover either the original computation or its result even if one has all the other information about the disguised computation. *Cost*: There is a cost to apply the disguise and a cost to invert it. The tradeoff between cost and security is application-dependent.

The ideal disguise is invertible, highly secure and cheap. We have found disguises for scientific computations that are invertible, quite secure and of reasonable cost (Atallah *et al.*, 2001). Not unexpectedly, we see that increasing security involves increasing the cost.

Random objects. The first class of atomic disguise techniques is to create random objects: numbers, vectors, matrices, functions, parameters, etc., which are "mixed into" the computation to disguise it. These objects are created from random numbers which, in turn, use random number generators. It is sufficient to save the seed and parameters of the generator.

Security can be achieved by taking a few standard generators (uniform, normal, etc.) and combining them, creating *one time random sequences*.

There are many standard probability densities (uniform, normal, exponential, etc.), with well honed pseudo-random number generator algorithms. Taking three of these, say G_1, G_2 and G_3 and combining this by

$$\alpha_1 G_1 + \alpha_2 G_2 + \alpha_3 G_3$$

where $\alpha_1 + \alpha_2 + \alpha_3 = 1$ provides a huge array of random number generators. However, there is no point in having the standard statistical densities, in disguise one merely wants numbers from a certain range. Thus, if one chooses G_1 as uniform on [0,a], G_2 as uniform on [0,$1-a$] and G_3 as uniform on [0,1], then one has a cheaply generated set of random number sequences with 3 independent parameters. Using 6 decimal digits for these parameters, provides 10^{18} distinct probability densities, sufficient to make one time pads for any near term application. These random numbers must be scaled appropriately for the computation to be disguised. Note that in creating this one set of random numbers, we use 7 sub-keys, the 1 parameter a, the 3 coefficients, and the 3 seeds of the generators.

Once one has random numbers then it is straightforward to create random vectors, matrices and discrete objects for use in disguises. We need to be able to create random sets of functions also. The technique to do this is as follows. Choose a basis of 10 or 30 functions for a high dimensional space F of functions. Then choose a random point in F to obtain a random function. The basis must be chosen with some care for this process to be useful. The functions must have high linear independence (otherwise the inversion process might be unstable) and their domains and ranges must be scaled compatibly with the computation to be disguised. One can make F a *one time random space* as follows: Enclose the domain of the computation in a box (interval, rectangle, box, ..., depending on the dimension). Choose a random rectangular grid in the box with 10 lines in each dimension and assuring a minimum separation (say 3%). Create K sets of random function values at all the grid points (including the boundaries), one set for each basis function desired. These values are to be in the desired range. Interpolate these values by cubic splines to create K basis functions. These functions are smooth (they have two continuous derivatives). Add to this set of K basis functions a basis for the quadratic polynomials.

This approach can be modified to make many kinds of one time random spaces of functions. The computational techniques for creating all these splines (or piecewise polynomials) are given in the book by deBoor (deBoor, 1978).

Symbolic Disguise with Identities. The paper (Atallah *et al.*, 2001) presents a lengthy set of techniques to disguise symbolic expressions using mathematical identities and partitions of unity. An automatic system is described which, for example, replaces x by

$$\begin{aligned}
&x^3 * sec^2(x) + x * (1-x) * sec^4(x) + \\
&x * tan^4(x) * cos^2(x) - x^2 * (1-x) * tan^2(x) * sec^2(x) + \\
&x^2 * (1-x) * sec^2(x) - x^3 * tan^2(x) * sec^2(x) - \\
&x * (1-x) * sec^2(x) * tan^2(x) + x * tan^4(x) * sin^2(x).
\end{aligned}$$

There is a rich area of mathematical function (Abramowitz and Stegun, 1964) which supports this technique and arbitrarily complex (and obsfucating) disguises can be created.

This approach can be extended to computer programs in general (one can view mathematical expressions as just programs for a very specialized "mathematics machine"). A program identity is a code that computes a known, simple value, e.g., 0 or 1. Then a program statement like: *If* $(k = 4)$ *then* $Y = Y + 10$ can be replaced by *If* $(1 * k = 4$ *and (not false)) then* $Y * 1 = Y + 0 + 10 * 1$. Next, one substitutes a 200 line program identity for *false*, three different 400 line program identities for 1, and a 300 line program identity for 0 into this statement for 1, false, 1, 0, and 1, respectively. This huge "expression" is expanded into a set of simple program steps which are then rearranged to entangle and obscure the simple statement. The resulting 1500 line program can be made intractable to simplify. Although we cannot prove it is "hard" to simplify such programs, evidence can be given that simplification is computationally intractable: Given a boolean formula

in disjunctive normal form (i.e., as a "sum of products") then it is NP complete to determine whether it is a tautology (i.e., can be simplified by replacing it with "true") — this is simply a re-statement of the textbook fact that satisfiability of a boolean formula in conjunctive normal form (i.e., as a "product of sums") is NP complete.

Other Atomic Disguise Techniques. The paper (Atallah *et al.*, 2001) presents several other techniques for disguising mathematical computations. These include *Modification of Linear Operators*: The functions (solutions, coefficients, etc.) in these are changed by disguises. *Manipulation of Objects*: Random functions are added or multiplied as appropriate. *Domain and Dimension Modification*: Domains can be extended by enlarging the definitions of functions. The dimension of a linear algebra problem can be increased or decreased by the customer. The domain can be split into several parts and then each of them disguised in a different way. *Coordinate System Changes*: Random changes can be done both for continuous problem (e.g., differential equations) or discrete ones (e.g., linear systems). Nonstandard coordinate systems can be used as effective, low cost disguises. *Data Dependent Disguises*: Have some of the random number generators depend on original problem data instead of the usual keys.

4 MUTUALLY SECURE FINGERPRINT MATCHING

Matching fingerprints is an important technology for biometric identification and we show how this matching can be disguised for outsourcing. Moreover, we show how it can be done by multiple parties using mutually secure outsourcing as done in Section 2. Fingerprint identification has three distinct levels: **Level 1**: The pattern of ridges are classified into one of a small number of types depending on the general shapes of the patterns. For each type, some parts (features) are identified, e.g., the center of a whorl or the points where a ridge separates into two branches. **Level 2**: Some simple counts are made using the point features, e.g., the number of ridges crossing a line between two points, or the number of ridges in a whorl. There are perhaps 4 to 20 possible values of these counts, depending on the fingerprint type and particular point features used. The types, plus the associated counts, generate a system with many thousands of classifications. However, real fingerprints are not evenly distributed among these classes and the chances of an accidental match can be as low as one in a thousand for the more common classes. **Level 3**: *Minutia* are small details such as: a ridge with a break, two ridges joined, a ridge has an "island" in it, a bridge (short ridge) joining two ridges. There are hundreds of minutia on a real fingerprint, many more than enough to make fingerprints unique among all the people in the world. A level 2 class is just a list of class type followed by the counts in a standard order. Two fingerprints are compared by matching these lists. The comparison is approximate because counts sometimes vary by one, due to effects in taking fingerprints.

Fingerprints can be disguised at Level 2 by mapping the image domain (a rectangle) by a random, smooth function. This preserves the classification while hiding the actual fingerprint into another rectangle. Then the resolution is decreased so that main ridges are clearly visible, but the minutia are not. Alternatively, disguised fingerprints can be compared by making several two-dimensional transforms of each image and comparing these. This adds greatly to the accuracy of identification when combined with the classification.

Fingerprints are compared at Level 3 by examining the minutia carefully (usually with magnification). Each minutia is typed and located on a portion of the image. Even a quarter of a fingerprint has about 100 minutia. Some of the minutia are innate to the person and others are the result of injuries, unclear fingers or fingerprinting devices, etc. Perhaps 20–40% of the minutia change over time, due to temporary and processing effects. However, this still provides ample basis to identify someone conclusively.

The minutia are represented as a network (planar graph) of nodes with types at each node. Subgraphs of these graphs can be represented as strings and then disguised. The string comparison method of (Atallah *et al.*, 2001) can be used to measure the similarity of the disguised subgraphs. Note than an accidental match of even 10% of two subgraphs is extremely unlikely and a 50% match is considered sufficient for an absolutely conclusive match.

The comparison of fingerprints can be outsourced with mutual security. Thus a person can have his fingerprint taken and disguised. Then it can be compared against a set of fingerprints of known persons disguised in the same way. The security can be enhanced by having a device that automatically applies the disguises to the actual fingerprints and then compares them without giving any external access to the fingerprints.

5 SECURITY ANALYSIS

The nature of disguises is that they may be broken completely (i.e., the disguise program is discovered) or, more likely, they are broken *approximately*. That is, one has ascertained with some level of uncertainty some or all of the objects in the original computation. There is a

probabilistic nature of breaking disguises which comes both from the use of random numbers and from the uncertainty about the disguise method. Of course, an attacker could only guess at the levels of certainty about the object information obtained. We discuss some attack strategies and possible defenses against them.

Statistical attacks. An attacker may attempt to derive information about the random number generators used. A determined attacker could attempt to check all the numbers in the outsourced computation against all the numbers particular random generators produce. (But note that this exhaustive match attack does not work if the random numbers we generated are added to the private data to disguise it – the data disguises them as much as they disguise it.) One cannot be complacent about the risk of a brute-force attack when we see teraflops or petaflops computers coming into use. Possible defenses against this kind of attack include (i) using random number generators with long (real and random) parameters; (ii) restarting the random number generators from time to time with new sub-keys (or change the random number generator used from time to time); (iii) using combinations of random number sequences; (iv) using data values to generate seeds for random number generators and, sometimes, replace randomly generated values by actual data values or other data dependent values.

Approximation theoretic attacks. The disguise functions are chosen from spaces described in Section 3. Let F be the space of these functions, $u(x)$ be an original function, $f(x)$ be a disguise function so that $g(x) = u(x) + f(x)$ is observable by the agent. The agent may evaluate $g(x)$ arbitrarily and, in particular, the agent might (if F were known) determine the best approximation $g^*(x)$ to $g(x)$ from F. Then the difference $g^*(x) - g(x)$ equals $u^*(x) - u(x)$ where $u^*(x)$ is the best approximation to $u(x)$ from F. Thus $g^*(x) - g(x)$ is entirely due to $u(x)$ and gives some information about $u(x)$. There are three defenses against this attack:

1. Choose F to have very good approximating power so that the size of $g^*(x) - g(x)$ is always small.

2. Choose F to be a *one time random space* as described in Section 3. Since F itself is then unknown, the approximation $g^*(x)$ cannot be computed accurately and any estimates of it must have considerable uncertainty.

3. Approximate the function object $u(x)$ by a high accuracy, variable breakpoint piecewise polynomial. It is known (deBoor and Rice, 1979) that this can be done efficiently and software exists to do this in low dimensions (deBoor, 1978). Then, one adds disguise functions with the same breakpoints and different values to the outsourced computation.

We conclude that if F has good approximation power and moderate dimension, then it is very hard to obtain any accurate information from the disguised functions.

Symbolic code analysis. Many scientific computations involve a substantial amount of symbolic input, either mathematical expressions or high level programming language (Fortran, C, etc.) code. It is natural to pass this code along to the agent in the outsourcing and this can compromise the security. An expression COS(ANGLE2 * x - SHIFT) + BSPLINE(A, x) is very likely to be the original function (COS $\cdots$) plus the disguise (BSPLINE $\cdots$) and they can be distinguished no matter how much the BSPLINE function values behave like COS as a function. The symbolic information may be pure mathematics or machine language or anything in between. Outsourcing machine language is usually impractical and, in any case, provides minimal security. Decompilers are able to reconstruct well over 90% of the original code from machine language. There are four general defenses against symbolic code analysis attacks:

1. Neuter the name information in the code. This means to delete all comments and to remove all information from variable names. This is an obvious, easy but important part of the defense.

2. Approximate the basic mathematical functions. The elementary built-in functions (sine, cosine, logarithm, absolute value, exponentiation, ...) of a language are implemented by library routines supplied by the compiler. There are many alternatives for these routines which can be used (with neutered names or in-line code) in place of the standard names. One can also generate *one time elementary function approximations* for these functions using a combination of a few random parameters along with best piecewise polynomial, variable breakpoint approximations. In this way all the familiar elementary mathematical operators besides arithmetic can be eliminated from the code.

3. Apply symbolic transformations and use of identities and expansions of unity. Changes of coordinates are very effective at disguise but can be expensive to implement. Consider the potential complications in changing coordinates in code with hundreds or thousands of lines. The other transformations are individually of moderate security value, but they can be used in almost unlimited combinations so that the combinatorial effects provide high security. For example, we transform the

simple differential equation

$$y'' + x * \cos(x)y' + (x^2 + \log(x))y = 1 + x^2$$

into

$$\begin{aligned}
& y''[x01 * x02(x) - x03] \\
+\ & y'[x04 * x/(x05\cos(x+1) + \cos x * x06(x)\tan(x+1)] * \\
& [x07 - \sin^2 x - x08(x)\sin^2 x + x07\sin^2(x+1)] \\
+\ & y[x01 * (x * x09(x))^2 - x10(x + \log x) + x11\cos x \log x^2] * \\
& [x12 * x13(x) + x14\tan x + (x15\sin x + x16\cos x + x17)] \\
=\ & \sin x + x18 * (1 + x^2) * x09(x) + x19(x) + x10 * x^2\cos x.
\end{aligned}$$

It certainly would take a considerable effort to recover the simple equation from this, and this disguise uses rather elementary techniques. A more secure disguise could use hundreds of lines.

4. Use reverse communication. This is a standard technique (Rice, 1993) to avoid passing source code into computations and can be used to hide parts of the original computation.

We conclude that symbolic disguises can be made as secure as one wants at the cost of increasing the complexity of the computation.

6 COST ANALYSIS

There are four components of the cost of disguise.

Computational cost for the customer. All of the disguise techniques proposed here and in (Atallah *et al.*, 2001) are affordable in the sense that the computation required of the customer is proportional to the size of the problem data. The computational cost of disguise tends to be lower than the cost of encryption. The principal cost of the customer is, in fact, not computational, but in dealing with the complex technology of making good disguises. A high level problem solving environment is needed to minimize this cost so an average scientist or engineer can quickly create disguises that provide complete security. For standard problems everything can be automatic.

Computational cost for the agent. The disguise techniques could substantially increase the computational cost for the agent. The effects are problem dependent and in the cases examined so far, disguise has a small effect on the agent's computational cost. None of these disguise techniques change the basic type of the computation.

Network costs. Disguises can increase network traffic by increasing (1) the number of data objects to be transmitted, and (2) the bulk of the data objects. A review of the disguises proposed so far shows that the number of objects is rarely changed much. However, the bulk of the individual objects might change significantly. Coordinate changes and the use of identities can change functions from expressions with 5–10 characters to ones with many dozens of characters. This increase is unlikely to be important in most applications; the size of the symbolic data is very small compared to the size of the computation. Disguise techniques for integers can increase their length by a bit or two which could increase their effective length from 1 byte to 2 bytes. This might double the network costs for some types of data intensive applications.

Cost summary. (1) Customer costs for disguise are reasonable and linear in the size of the computation data. The principal cost is in the "intellectual" effort needed to make good disguises. (2) Agent costs have a minimal increase unless the customer changes the problem structure. The control of problem structure increases the intellectual effort of the customer. There might be especially simple computations where good disguise requires changing the problem structure. (3) Network cost increases vary from none to modest, rarely will this cost be doubled.

References

Abramowitz, M., and Stegun, I.A. (1964) *Handbook of Mathematical Functions*. Appl. Math. Series 55, Nat. Bur. Stnds., U.S. Govt. Printing Office.

Atallah, M.J., Pantazopoulos, K.N., Rice, J.R. and Spafford, E.H. (2001) Secure outsourcing of scientific computations. In *Advances in Computers*, Vol. XX (Z. Zelkowitz, ed.), Academic Press, 56 pages, 2001, to appear.

Collberg, C., Thomborson, C., and Low, D. (1988) *A taxonomy of obfuscating transformations*. Tech. Rpt. 148, Department Computer Science, University of Auckland.

deBoor, C., and Rice, J.R. (1979) An adaptive algorithm for multivariate approximation giving optimal convergence rates. *J. Approx. Theory*, **25**, pp. 337–359.

deBoor, C. (1978) *A Practical Guide to Splines*. SIAM Publications.

Rice, J.R. (1993) *Numerical Methods, Software, and Analysis*. Second Edition, Academic Press, Section 7.6.D.

Schneier, B. (1996), *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition, John Wiley & Sons, Inc..