Provably Secure Data Hiding and Tamper Resistance for a Simple Loop Program*

Rida A. Bazzi† K. Selçuk Candan Raphael Badin‡ Aziz Fajri§
Computer Science Dept.
Arizona State University
Tempe, AZ, 85287

ABSTRACT

We study the problem of computing with encrypted data. We propose a hiding scheme that allows a client to execute a simple loop program with real or complex inputs securely on a server. This is the first hiding scheme that we are aware of that applies to real and complex data. The scheme allows the client to efficiently determine with high probability whether the results returned by the server are correct. The scheme we propose uses new techniques that have not been used previously in this context.

Keywords: encryption, hiding, security, tamper resistance.

1. INTRODUCTION

Computing with encrypted data is desirable for many applications. In this paper, we are interested in the problem of non-interactive computation outsourcing. In this problem, a client uses a server to execute a program. The client provides the program and the input data and the server provides the computational resources. We are interested in outsourcing the computation while leaking minimal information about the input data to the server. Also, the solution should be non-interactive in that the client and server exchange only two message: one to send the original input to the server and one to send the results to the client. A desirable feature for a solution to this problem is tamper resistance or robustness. This refers to the ability of the client to efficiently test whether the results returned by the server are correct.

Work on computing with encrypted data or code uses many models of interaction that can differ in subtle ways. For example, the client can interact with one or many servers and the interaction can consist of a constant number or a variable number of rounds. Also, the assumptions about the computation powers of the client and server can vary. Some work assumes that the server is an oracle with unlimited computation power and that the client can recover the results of the computation in polynomial time.[1]

The rest of this paper is organized as follows. Section 2 presents the problem we are addressing and the interaction model. Section 3 highlights our contributions. Section 4 describes related work. Section 5 formally defines hiding and introduces our execution model. Section 6 presents our techniques for hiding simple loop programs. Section 7 presents techniques to check the correctness of the returned results. Section 8 concludes the paper.

2. MODEL

In computation outsourcing, a client has input data X for which it wants to calculate the value of $P(X)$, where P is a program with X as input. In order to save on computation, the client wants a server that it does not trust to calculate $Y = P(X)$ in such a way that the server learns nothing about X from the computation other than what it already knew about X. To that end the client will present an encrypted input to the server. The server will apply a, possibly encrypted, program to X and produces a result Y' that it sends to the client. The client

*This work is supported in part by the Air Force Office of Scientific Research under grant F49620-00-1-0063.
†The work of this author is supported in part by the National Science Foundation CARRER award CCR-9876052.
‡Contributed to the work while visiting ASU, Summer 2000. Current affiliation not available
§Contributed to the work while visiting ASU, Summer 2000 . Current affiliation not available

Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Defense and Law Enforcement II, Edward M. Carapezza, Editor, Proceedings of SPIE Vol. 5071 (2003) © 2003 SPIE · 0277-786X/03/$15.00

should be able to decrypt Y' to obtain the output $Y = P(X)$ - computation outsourcing. The client should also be able to tell with high probability that the output it calculates is correct - robustness. In order for such a scheme to be useful, it should provide computation savings to the client. The overhead of the scheme can be divided into the following parts. First, there is the time of encrypting X and decrypting Y' which should be smaller than the time to calculate $P(X)$ by the client. Second, if robustness is provided, then the time to encrypt X and decrypt Y' added to the time needed to determine that the result is correct must be smaller than the time needed to calculate $P(X)$ by the client.

In addition, in schemes that require the program to be encrypted, there is the time to encrypt the program. In some schemes, such encryption is done only once, in which case that cost is an *initialization* cost. In general, we will not be concerned with initialization cost, but we will describe it for our schemes.

We require our schemes to be non-interactive. This means that the client sends only one message to the server and the server replies with only one message to the client. Non-interactive schemes are desirable because they have low communication overhead.

Another overhead introduced by hiding schemes are message size and memory overhead. The size of the messages exchanged by the client and server should be small. Also, a hiding scheme should not introduce a large memory overhead at the client side.

In our model, we are not much interested in the overhead at the server side as we assume that the server has large computation resources.

In our setting, we are only interested in hiding the data, so we assume that the server has access to the program and the encrypted data but not to the original data. Also, the client will be generating some local random keys used in encryption. Those keys are private to the client and are not communicated to the server.

3. RESULTS

We propose a non-interactive scheme to provide data hiding for a program of the form

for l **do** $X = MX$

where X is a complex input vector over C^n, l is an variable integer input, and M is a $n \times n$ complex matrix. The scheme allows a client to encrypt X without leaking any information about it to the server. Unlike other proposed schemes, our scheme allows a client to determine efficiently and with high probability whether the results returned by the server are correct or not. The scheme is non-interactive; the client sends only one message to the server and the server sends only one message to the client. This is the first hiding scheme that we are aware of that apply to real and complex inputs. Other hiding schemes typically assume that the inputs are elements of Z/mZ which reduces their applicability. While our results apply to a simple class of loop programs, the techniques we develop are interesting on their own.

The encryption cost of the scheme is of the order n^2 and is independent of l[¶]. The execution time of the program is ln^2 or $T(n)\lg l$, where $T(n)$ is the execution time of an algorithm to square the matrix M. The time ln^2 is obtained by applying a straightforward execution of l iteration each requiring n^2 operations. The time $T(n)\lg l$ is obtained by calculating M^l and then multiplying the result with X. If l is large, in both cases there are large saving in computation for the client.

It should be clear that a simple way to hide the input for our program is to have the server calculate M^l and send the result to the client. This has two drawbacks. First, the communication complexity is n^2 to send a matrix instead of n to send a vector. Second, it is not clear in that case how the checking for correctness can be achieved.

Another interesting problem that can be solved with our scheme is that of joint computation between the client and server. In that problem, there is a function f with two inputs that the client and server want to calculate. One input is provided by the client and one by the server. The calculation should be done is such

[¶]We will discuss initialization costs in Section 8

a way that the client and serevr learn nothing about the other's input except what they can deduce from the computed value. In our setting, if l is provided by the server, then the client and server to compute $f(X, l) = M^l X$ without leaking any information the client or server other than what they can deduce from the value of $M^l X$. This setting was also considered by other researchers.[10] This alternative view of the problem further rules out the the simple solution of having the server send M^l to the client.

An important difference between our result and other data hiding results is that our results apply to plynomial-time programs. Other results apply to problems not known to have polynomial-time solutions. Further discussion of the efficiency of the scheme is provided in Section8.

4. RELATED WORK

Related works differs greatly in the assumption they make. Sander and Tschudin[9] consider the problem of code hiding and use homomorphic encryption functions to hide polynomials. Their encryption schemes are efficient, but their scheme has some major weaknesses. For instance, the host can tell which coefficients are equal to zero. Also, if two coefficients are equal, then the corresponding encrypted coefficients will also be equal. Furthermore, they do not consider the statistical properties of the coefficients in their solution, and their argument for the security of their solution is informal. Formal theoretical work on input hiding was done by others.[1,7] Feigenbaum[7] considers the problem of encrypting problem instances. In her work, she attempts to hide the input to a function, but does not hide the function itself. Also, she does not give a formal definition of what hiding means. The results of Feigenbaum[7] were later refined[1] and formal definitions of hiding and information leakage is proposed. We generalize those definitions in this paper. In the previous definitions,[1] it is assumed that the owner of the code has polynomial-time computing power and that the host is an oracle with unlimited computing power. This leads to some hiding solutions for some problems in which the owner hide polynomial-time functions by calculating the answer locally. This is particularly appealing from a complexity point of view, but the approach foes not satisfactorily address the problem of instance hiding for problems that have polynomial-time solutions. In some other work,[3] a general hiding scheme for boolean functions is presented. Unfortunately, the scheme uses a standard representation of boolean function that can introduce an exponential overhead, which makes it impractical. Also, the scheme only applies to boolean functions and not to general functions. A drawback of the models we describe above[1,3] and models used by other researchers is that they allow a polynomial number of communication rounds between the host and the owner of the code. A different approach is taken by Schneider,[11] where it is assumed that up to t machines in the system are malicious. Faulty platforms are detected by replicating the agent computation and executing them on different hosts. Results are obtained by collecting the results and the use of voting. The solution uses interesting encryption techniques to make sure that faulty machines cannot collude to spoof results. The work of Schneider[11] does not consider techniques for providing tamper resistance. Sander et. al[10] solve a problem related to data and code hiding. In their problem, the client has a private input x and the server party has a private function f. The goal is to calculate $f(x)$ non-interactively without leaking any information about x to the server or f to the client, other than what can be inferred from the value of $f(x)$. They provide a check of correctness that requires the client to provide the server with $3l$ inputs (most of them with known output values) to reduce the probability of successful cheating to 3^{-l}. The call their technique witness-based function checking.

Aucsmith[2] proposed interesting code obfuscation techniques, but does not study formally their security properties. Work on code obfuscation tends to be informal and ad hoc and lacks the rigor that our approach presents. There is a lot of work on code obfuscation,[4–6] but we do not discuss it because it is not relevant to *provable* code hiding and tamper resistance techniques.

5. CODE AND DATA HIDING

5.1. Execution Model

To execute a program P on an input x, the client sends the encrypted program P' and the encrypted input x' to the remote host (server). The host then sends the encrypted output $y' = execution(p', x')$ to the client and the client decrypts y' to obtain $y = execution(p, x)$. In our model, the client knows all the information about the input and the program. We adopt a standard information-theoretic model for hiding. We model the

adversary's knowledge about the client's program/input as a probability space over the set Π/E of all possible program/input pairs. This space is defined by a probability distribution. An input is a vector of real or complex values and a program is a string from a set of valid strings. In general, we would expect that the probability distribution be equal to zero for most program/input pairs and only a subset of inputs and programs would have a positive probability distribution.

In our results, we only provide input hiding. Therefore, when we prove that our scheme does not leak information about the input, we assume that the server knows the source program, the encrypted program and the encrypted input. The hiding guarantees we provide apply even if the server has access to an unbounded number of encrypted inputs.

We assume that the client has access to a cheap source of randomness that produces real numbers according to the normal distribution. In practice, data is discrete and an approximation of this source can be used.

Without loss of generality we assume that the inputs and outputs have the same domain E. Let Π be the set of all possible programs under consideration (with inputs and outputs in E). We use the ' symbol to denote the encrypted domains. So E' is the set of encrypted inputs and outputs and Π' is the set of encrypted programs. In our definition, encryption generates keys that are used in decryption. We denote the key space by K. In some schemes, we need to generate two keys, one for the code (program) and one for the data (input or output). We denote the data key space by K_d and the code key space by K_c.

We are interested in developing hiding schemes that leak no information about the input. We express information about the input as a property of the input, which is simply a function of the input.

5.2. Hiding function

DEFINITION 5.1. *A* computation hiding function *is a randomized function* $f : \Pi \times E \rightarrow \Pi' \times E' \times K$ *such that there exists a decryption function* $df : E' \times K \rightarrow E$ *such that*

$$\forall (x, p) \in E \times \Pi \quad y = df(y', k),$$

where $(p', x', k) = f(p, x)$, $y = p(x)$, *and* $y' = p'(x')$.

In the definition, we explicitly model the private key k that can be used in the decryption. The host only sends p' and x' to the server.

The definition of a computation hiding function allows for the program to be encrypted differently for different inputs. Hiding functions in which the program is encrypted the same way for all inputs are of special interest. We call such functions *separable computation hiding functions*. Separable hiding functions are best described with two hiding functions, one for code and one for data.

DEFINITION 5.2. *A* code hiding function *is a randomized function* $f_c : \Pi \rightarrow \Pi' \times K_c$.

DEFINITION 5.3. *A* data hiding function *associated with a code hiding function* f_c *is a randomized function* $f_d : \Pi \times E \times K_c \rightarrow E' \times K_d$ *such that there exist a decryption function* $df : E' \times K_d \rightarrow E$ *such that*

$$\forall (x, p) \in E \times \Pi \quad y = df(y', k_d),$$

where $(p', k_c) = f_c(p)$, $(x', k_d) = f(p, x, k_c)$, $y = p(x)$, *and* $y' = p'(x')$. The pair (f_c, f_d) naturally defines a separable computation hiding function, so we abuse notation and write $f = (f_c, f_d)$.

We are interested in hiding functions that leak little or no information about the code or input.

DEFINITION 5.4. *A computation hiding function* f leaks at most property $\wp$ *of input* x *for a given program* p *if* x *is independent of* (x', p') *relative to* $\wp$. *In other words the conditional probability of* x *given* $\wp$ *is equal to the conditional probability of* x *given* $\wp$, *and* (x', p').

We are interested interested in hiding functions that do not leak information about the input even if the server has many encrypted input values.

DEFINITION 5.5. *A computation hiding function* f *that* leaks at most property $\wp$ *of input* x *for a given program* p *has memory* m *if* x *is independent of any sequence* $(x'_1, p^{1\prime}), (x'_2, p^{2\prime}), \ldots (x'_1, p^{m\prime})$ *relative to* $\wp$, *where* $x_1, \ldots, x_m$

are the encyprions of inputs $x_1, \ldots, x_m$ *and* $p^{1'}, \ldots p^{m'}$ *are the corresponding encryptions of program* p. Similarly we can define what it means for a computation hiding function leak a at most a property $\wp$ of a program. The hiding scheme we present has infinite memory.

5.3. Correctness Check

We are interested in hiding functions that enable the client to efficiently determine whether the returned results are correct or not. While it is always possible to determine the correctness of the result by executing the program on the client, this clearly is not efficient. In order to test the correctness of the result efficiently we allow the test to be incorrect with small probability.

DEFINITION 5.6. *A* check for correctness *associated with a hiding function* f *and a program* p *is a randomized function* $C_{(f,p)} : E \times E' \times K \rightarrow \{true, false\}$

For the case where f is separable, K is replaced with $K_c \times K_d$ in the mapping above. We are interested in checks of correctness that are accurate.

DEFINITION 5.7. *A check of correctness is accurate with accuracy* a *if* $\forall(x,p)\ C_{(f,p)}(x, p'(x')) = true$ *and* $pr(y' \neq p'(x') \mid C_{(f,p)}(x,y') = true) < a)$. Note that in the definition of accuracy, both the hiding function (which is randomized) and the check for correctness (which is also randomized) contribute to the probability.

6. HIDING SCHEME FOR A SIMPLE LOOP LINEAR PROGRAM

A simple linear loop program $P(x,l)$ is a program of the form

for l **do** $X = MX$

where X is a vector of n dimensions and M is a $n \times n$ matrix. The parameter l is the number of iterations of the **for** loop. The output of the program is $M^l.X$. We assume that X is independent of M and l. In other words $P(X \mid M \cap l) = P(X)$.

In this section we present a hiding scheme that leaks at most the size of the input X and with memory ∞. The scheme is complicated, so we start by presenting an outline.

The idea of the scheme is simple. Instead of sending X to the server, the client sends X' such that $X' = A.X$, where A is an invertible matrix that commutes with M. Instead of calculating $Y = M^l X$, the server calculates $Y' = M^l.X'$. So, $Y' = M^l.A.X = A.Y$. The client can recover Y by multiplying Y' with A^{-1}. The goal is to hide X by multiplying it with A, so that the server cannot learn anything about X from the value of X'.

The simple idea of the scheme does not work for all input values. The simple idea only works for inputs such that $P.X$ contains no zero entries, where P is the Jordan transformation matrix of M ($P^{-1}.M.P$ is the Jordan canonical form of M). For inputs such that $P.X$ contains zero entries, we change the scheme so that the input X is split into two input Y and Z such that $P.X$ and $P.Y$ contain no zero entries.

Also, the simple idea will leak some information about the input, namely the maximum of the ratios of pairs of entries of $P.X$. Again, we change the scheme so that the input X is split into two inputs Y and Z each with a maximum ratio equal to 2 independently of X.

In proving that the scheme does not leak any information about the input X, we assume that the server knows M, l, X', and the form of the program.

As a first step, our goal is to prove the following theorem.

THEOREM 6.1. $pr(X \mid M \cap l \cap X')) = pr(X)$. In other words, the server does not learn any information about X given the values of M, l, and X', which is all the information it has available. Since M, l and X are independent, the theorem is equivalent to:

THEOREM 6.2. $pr(X \mid X') = pr(X)$. This theorem does not prove that the hiding scheme has infinite memory. Our real goal is to prove the following theorem.

THEOREM 6.3. *$pr(X \mid M \cap l \cap X' \cap X'_1 \cap \ldots \cap X'_m)) = pr(X)$, where $X'_1, \ldots, X'_m$ is an arbitrary sequence of encrypted inputs.* In other words, the server does not learn any information that it does not already know about X given M, l, the encrypted input X' and any number of encrypted inputs. Given the independence of M, l and X, the theorem is equivalent to

THEOREM 6.4. *$pr(X \mid X' \cap X'_1 \cap \ldots \cap X'_m)) = pr(X)$, where $X'_1, \ldots, X'_m$ is an arbitrary sequence of encrypted inputs.*

6.1. Commuting Matrices

The encryption scheme uses a matrix A that commutes with M. The set of matrices commuting with M can be expressed as a function of M using the Jordan normal form of M. We start by recalling the definitions of Jordan normal form and Toeplitz matrices.

Let M be a matrix whose coefficients are in C. Then there exists an invertible matrix P such that $M = P.J.P^{-1}$ where $J = \begin{pmatrix} J_1 & & \\ & \ddots & \\ & & J_p \end{pmatrix}$ and $\forall s \in [1,p]$ $J_i = \begin{pmatrix} \lambda_s & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_s \end{pmatrix}$.

The numbers $\lambda_1, ..., \lambda_p$ are the eigenvalues of M. The matrix $J_s \in C^{k_s \times k_s}$ is the Jordan block of the Matrix M corresponding to the eigenvalue λ_s. The only non-zero entries of J belong to the Jordan blocks. We call the matrix P the *Jordan transformation matrix* of M.

An upper-triangular Toeplitz matrix $A \in C^{n \times n}$ is a matrix of the form:

$$A = \begin{bmatrix} \alpha_0 & \cdots & \alpha_{n-2} & \alpha_{n-1} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \alpha_0 \end{bmatrix}$$

We note that the product of two upper triangular Toeplitz matrices of the same size is also a Toeplitz matrix. Also upper triangular Toeplitz matrices of the same size commute.

Now we are ready to give the statement of the Commuting Theorem.

THEOREM 6.5.[8] *If $M \in C^{n \times n}$ and $M = P.J.P^{-1}$ where J is the Jordan canonical form of M, then a matrix A commutes with M if and only if it is of the form $A = P.Y.P^{-1}$ where $Y = [Y_{st}], 1 \leq s,t \leq p$, is matrix that is consistent with the partition of J into Jordan blocks, and where $Y_{st} = 0$ for $\lambda_s \neq \lambda_t$ and Y_{st} is of the forms (a), (b), or (c) if $\lambda_s = \lambda_t$.*

(a) if $k_s = k_t$ then Y_{st} is an upper-triangular Toeplitz matrix of size $k_s = k_t$.

(b) if $k_s < k_t$ then $Y_{st} = \begin{bmatrix} 0 & Y_{k_s} \end{bmatrix}$

(c) if $k_s > k_t$ then $Y_{st} = \begin{bmatrix} Y_{k_t} \\ 0 \end{bmatrix}$, where Y_{k_s} and Y_{k_t} are upper triangular Toeplitz matrices of size k_s and k_t respectively.

The commuting theorem gives the general form of an invertible matrix A commuting with M. For our purposes, we restrict our attention to matrices of the form $A = P.J_A.P^{-1}$, where $J_A = [J_A^{st}], 1 \leq s,t \leq p$, and J_A^{ss} is an upper triangular Toeplitz matrix of size k_s and $J_A^{st} = 0$ if $s \neq t$.

6.2. Choosing A

In this section we show how to chose A so that X and X' are independent.

The proof is in three main steps. First we assume that $P.X$ has no zero coefficients, where P is such that $M = P.J.P^{-1}$ and J is the Jordan form of M. We show that A can be chosen according to a particular probability distribution so that the distribution of X' depends minimally on X. Second, we show how to split the input to handle cases where $P.X$ has zero entries. Third, we show how to split the input so that the probability distribution of X' is completely independent of X.

The first step is to find a distribution of A that guarantees that X' is minimally dependent on X (in a sense that will become clear below). We start by recalling properties of normal distributions and introduce quasi-normal distributions.

A well-known result about normal distributions is that the linear combination of random variables following different normal distributions is a random variable following a normal distribution whose parameter can be expressed as a linear combination of the parameters of the original distribution:

$$\begin{cases} \alpha_1 \sim \aleph(m_1, \sigma_1^2) \\ \alpha_2 \sim \aleph(m_2, \sigma_2^2) \\ \quad\vdots \\ \alpha_n \sim \aleph(m_n, \sigma_n^2) \end{cases} \quad \Rightarrow \quad \sum_{i=1}^{n} a_i.\alpha_i \sim \aleph(\sum_{i=1}^{n} a_i.m_i, \sum_{i=1}^{n} a_i^2.\sigma_i^2),$$

where $\sim$ denotes "follows the law" and $\aleph(m, \sigma^2)$ denotes a normal distribution with mean m and standard deviation σ.

In our results we need to use random variables that follow a modified normal distribution in which the value zero is never generated. We denote such a distribution with $\aleph_0(m, \sigma)$ and we call it quasi-normal distribution. As for normal distributions, the linear combination of random variables following different quasi-normal distributions is a random variable following a quasi-normal distribution.

We know that A is of the form $A = P^{-1}.J_A.P$ where J_A has the same shape as the Jordan normal form of M. Without loss of generality, we assume that J_A has only one block. It follows that J_A is of the form:

$$J_A = \begin{bmatrix} \alpha_0 & \cdots & \alpha_{n-2} & \alpha_{n-1} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \alpha_0 \end{bmatrix}$$

So, choosing A reduces to choosing α_i.

We know that $X' = A.X$ so $(P.X') = J_A.(P.X)$. Let $P.X' = (\gamma_i)_{0 \le i \le n-1}$ and $P.X = (\delta_i)_{0 \le i \le n-1}$, and let the function $maxr$ be defined as follows:

$$maxr(X) = Max\{\, |\frac{\delta_i}{\delta_j}| \;:\; 0 \le i, j \le n-1 \}$$

Notice that this maximum exists since $\delta_i \neq 0$, $0 \le i \le n-1$. Let $max \ge maxr(X)$.

We show how the probability distributions of α_i can be chosen so that the distributions of γ_i do not depend on the coefficients δ_i but only on max as follows:

$$\begin{cases} \gamma_{n-1} \sim \aleph_0(0, 1) \\ \gamma_{n-2} \sim \aleph_0(0, max^2) \\ \quad\vdots \\ \gamma_0 \sim \aleph_0(0, n.max^2) \end{cases}$$

By the relationship between $P.X$ and $P.X'$ above, we have:

$$\begin{cases} \gamma_{n-1} = \alpha_0.\delta_{n-1} \\ \gamma_{n-2} = \alpha_0.\delta_{n-2} + \alpha_1.\delta_{n-1} \\ \\ \gamma_0 = \alpha_0.\delta_0 + \cdots + \alpha_{n-1}.\delta_{n-1} \end{cases}$$

Let $\aleph_0(0, \sigma_i^2)$ be the distribution of α_i, $0 \leq i \leq n-1$. If we choose the coefficients σ_i recursively as follows :

$$\left\{ \begin{array}{c} \sigma_0^2 = \frac{1}{\delta_{n-1}^2} \\ \sigma_i^2 = \frac{1}{\delta_{n-1}^2}(i.max^2 - (\sum_{k=0}^{i-1} \sigma_k^2.\delta_{n-i+k-1}^2)) \quad \forall i \in [1, n-1] \end{array} \right\},$$

it follows that $\forall i \in [1, n-1]$, $\sigma_i^2 \geq 0$ (due to the choice of max) and that the entries of $P.X'$ follow the laws $\aleph_0(0,1)$, $\aleph_0(0, max^2)...\aleph_0(0, n.max^2)$ (by the properties of quasi-normal distributions) (we omit the details of the calculation).

So far, we have only proved that distribution of $P.X'$ only depends on an upper bound on $maxr(X)$. Now, we show how the input can be split into two inputs Y_m and Z_m such that $max = 2$ is acceptable upper bound for $maxr(Y_m)$ and $maxr(Z_m)$. The input X is the sum of Y_m and Z_m and $P.Y_m'$ and $P.Z_m'$ are completely independent of X and of each other. We first show how the input can be split to handle the case where $P.X$ has some zero entries. Then, we show how to split the input so that $max = 2$ is an acceptable upper bound.

6.2.1. Splitting the Input

We first show how to split X into Y and Z such that $X = Y + Z$ and $P.Y$ and $P.Z$ have no zero entries. Then we show how to split X into Y_m and Z_m such that $maxr(Y_m) \leq 2$ and $maxr(Z_m) \leq 2$. Encryption is applied independently to Y_m and Z_m without leaking any information about X and the output for X is recovered by adding the encrypted outputs for Y_m and Z_m. To avoid giving the server any additional information, we split X even if $P.X$ has no zero entries and $maxr(X) \leq 2$.

While many choices for Y and Z would work, we are interested in values for Y and Z that are easy to calculate and can lead to a simple correctness argument. We choose $Y = P^{-1}.(|P.X| + 1_v)$ where $|P.X|$ is a vector whose values are the absolute values of the vector $P.X$ and 1_v is a vector of size n whose entries are all equal to 1. We choose $Z = X - Y = P^{-1}.(P.X - |P.X| - 1_v)$. It is straightforward to verify that $P.Y$ and $P.Z$ have no zero entries.

Let $Y_{max} = Max\{|(P.Y)_i| : 1 \leq i \leq n\}$ and Let $Z_{max} = Max\{|(P.Z)_i| : 1 \leq i \leq n\}$ and Let max_v be a vector whose entries are all equal to $max(Y_{max}, Z_{max}) + 1$. Now consider the two vectors $Y_m = Y + P^{-1}max_v$ and $Z_m = Z - P^{-1}max_v$. Note that $X = Y + Z = Y_m + Z_m$. It is straightforward to show that $maxr(Y_m) \leq 2$ and $maxr(Z_m) \leq 2$. Also, both $P.Y_m$ and $P.Z_m$ have no zero entries.

Since $P.Y_m$ and $P.Z_m$ have no zero entries, we can encrypt Y_m and Z_m by multiplying them with A_y and A_z, where A_y and A_z are chosen independently of each other according to the distribution of A above. Since $maxr(Y_m) \leq 2$ and $maxr(Z_m) \leq 2$, 2 can be used in place of max in the equations.

Also, $pr(Y_m' \cap Z_m') = pr(Y_m').pr(Z_m')$ because the matrices used to hide Y_m ad Z_m are independent. This will be sufficient to prove that X is hidden by the scheme.

THEOREM 6.6. $pr(X \mid Y_m' \cap Z_m' \cap l \cap M \cap l) = pr(X)$

Proof. Since X is independent of M and l, it is enough to prove that $pr(X \mid Y_m' \cap Z_m') = pr(X)$ $pr(X \mid Y_m' \cap Z_m') = \frac{pr(Y_m' \cap Z_m' \mid X) pr(x)}{pr(Y_m' \cap Z_m')}$. But, by the choice of A_y and A_z $pr(Y_m' \cap Z_m' \mid X) = pr(Y_m' \cap Z_m')$. So, the result holds. □

The proof that X is independent of any number of encrypted split inputs is identical to the above proof as long as the encryption matrices of different inputs are chosen independently.

In the next section, we show how the client can efficiently determine whether the results returned by the server are correct or not. Given the results for input splitting, we assume that the input X is such that $P.X$ has no zero entries.

7. CHECKING FOR CORRECTNESS

Intuitively, the idea of checking for correctness is to make the server do some additional calculation that are mized with the program calculation and whose result the client knows beforehand. The calculation are mized in such a way that the server cannot return incorrect output without that showing in the additional calculation whose result the client knows.

7.1. Padding

In order to check for correctness, the matrix M needs to be padded. So, instead of providing the server with M, the server is provided with the encrypted form of a padded matrix P_M:

$$P_M = \begin{bmatrix} M & 0 \\ P_1 & p_2 \end{bmatrix}$$

where P_1 is a $1 \times n$ matrix and p_2 is a 1×1 invertible matrix both unknown to the server and appropriately chosen as explained below[‖].

The encrypted matrix P'_M is

$$P'_M = K^{-1}.P_M.K$$

where K is an invertible matrix chosen of the form $\begin{bmatrix} Id_n & 0 \\ K_3 & k_4 \end{bmatrix}$ to simplify calculation.

The choice of K implies that

$$P'_M = \begin{bmatrix} M & 0 \\ M_3 & m_4 \end{bmatrix},$$

where M_3 is a $1 \times n$ matrix and m_3 is a 1×1 matrix.

The input X is also padded and we write $X = \begin{bmatrix} X_1 \\ x_2 \end{bmatrix}$ where X_1 is the input vector of dimension n and x_2 is a scalar padding value.

By definition of K, we have:

$$\begin{bmatrix} Id_n & 0 \\ K_3 & k_4 \end{bmatrix} . \begin{bmatrix} M & 0 \\ M_3 & m_4 \end{bmatrix} = \begin{bmatrix} M & 0 \\ P_1 & p_2 \end{bmatrix} . \begin{bmatrix} Id_n & 0 \\ K_3 & k_4 \end{bmatrix} \quad (1)$$

The matrix equation is equivalent to the system of equations:

$$\begin{cases} K_3.M + k_4.M_3 = P_1 + p_2.K_3 \\ k_4.m_4 = p_2.k_4 \end{cases} \quad (2)$$

the two equations of system (2) are equivalent to,

$$\begin{cases} P_1 = K_3.M + k_4.M_3 - m_4.K_3 \\ p_2 = m_4 \end{cases}$$

since k_4 is invertible.

Hence, for all K_3, and invertible k_4 there exists P_1 such that relation (1) is satisfied. In what follows, we assume that all entries in K_3 and k_4 are uniform random variable in the interval $]0,1[$ and that P_1 is chosen accordingly. $(K3, k_4)$ is an element of the open hypercube $]0,1[^{n+1}$ with all coordinates chosen uniformly at random.

[‖]Notice that padding does not require much extra computations by the server

The correctness check is done in more than one step. First we assume that the client knows the correct value of x_2 then we show how the check of correctness can be done without that assumption.

We first note that padded part of the output should satisfy

$$x_2 = K_3.X'_1 + k_4.x'_2 \tag{3}$$

(here x_2 is the padded value of the output and $[X'_1, x'_2]^T$ is the encrypted output. If the server returns incorrect values for X', say $X'_1 + X'_{1\delta}$ and $x'_2 + x'_{2\delta}$, then $X'_{1\delta}$ and $x'_{2\delta}$ should satisfy:

$$K_3.X'_{1\delta} + k_4.x'_{2\delta} = 0 \tag{4}$$

so that the clients detects no modification of the value of x_2. any entry of $X'_{1\delta}$ and $x'_{2\delta}$ is not zero, then Equation (4) defines a part of a hyperplane inside the open hypercube $]0,1[^{n+1}$, where the non-zero values of $X'_{1\delta}$ and $x'_{2\delta}$ are the coefficients and the entries of K_3 and k_4 are the unkowns. In the continuous case, the probability that (K_3, k_4) belongs to that [art of hyperplane is zero. In practice, all value are discreet and we need to make the argument that the probability that the host can cheat without being detected can be made as small as needed. By a continuity argument, the accuracy over $]0,1[$ can be chosen so that the probability of Equation 4 being satisfied with non zero $X'_{1\delta}$ and $x'_{2\delta}$ can be made arbitrarily small.

The discussion so far assumed that the client knows the output value for x_2. Unfortunately, the client cannot compute the output for x_2 efficiently. So, the client has to make the server calculate that value for her! This can be achieved as follows.

We decompose the original input** (including the padded part) into two parts:

$$\begin{pmatrix} X_1 \\ x_2 \end{pmatrix} = \alpha. \begin{pmatrix} S_1 \\ s_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \beta \end{pmatrix}$$

with α and β being uniform random variables over $]0,1[$. The host is provided with $\begin{pmatrix} S_1 \\ s_2 \end{pmatrix}$ and $\begin{pmatrix} X_1 \\ x_2 \end{pmatrix}$ after encrypting them. Note that the output for $\begin{pmatrix} 0 \\ \beta \end{pmatrix}$ is easily computed and is equal to $\begin{pmatrix} 0 \\ p_2^l.\beta \end{pmatrix}$.

If the results returned by the server are $\begin{pmatrix} T_1 \\ t_2 \end{pmatrix}$ and $\begin{pmatrix} Y_1 \\ y_2 \end{pmatrix}$, the client checks if

$$\alpha.t_2 + p_2^l.\beta = y_2. \tag{5}$$

is satisfied. If the equality is satisfied, then the client can use the value of $\alpha.t_2 + \beta^n$ as the correct value of x_2 output. Indeed, the probability that the host returns incorrect values can be made arbitrarily small in this case also using an argument similar to the one used above.

The correctness check might seem too complicated. Unfortunately, we were not able to find any simpler check. The source of the problem is the following. The program we consider calculates a linear function of X. If the client sends X and $\alpha_1 X, \alpha_2 X, \ldots, \alpha_t X$, to the server, where α_i are random values, the server would have to return output values that obey the same ratios as the input values. Nevertheless, the server can still cheat by multiplying all the output values with a constant factor. Our approach does not suffer from this weakness.

**By original input we mean the input obtained after the splitting operation

8. DISCUSSION

The method requires that the calculation be done on 4 encrypted inputs instead of one: first by splitting the input into 2 and then by splitting again into 2 for the checking for correctness. This is a constant factor of the original execution time of the program (assuming the matrix M is not sparse). The encryption of the input requires $O(n^2)$ operations in addition to the generation of n random numbers according to normal distributions. The generation of the random numbers can be done by first generating a random number according to $\aleph_0(0,1)$ and then adding and multiplying by appropriate constants.

As far as we can tell, the execution time of the program is either ln^2 for a direct execution of l iterations or $T(n)\lg l$, where $T(n)$ is the time to square the matrix M. In both cases, the encryption time is smaller than the execution time for large l. In fact, in our model, we assume l to be provided as input to the server, which precludes any precomputation of M^l whether at the server or the client site. If the number of iterations we allow is fixed, then there is no point in the encryption because the cost becomes comparable to the program execution (for a fixed loop index l, we can precompute M^l).

The encryption requires the client to know the Jordan form for the program. Calculating the Jordan form can be expensive. In fact, once the Jordan form is known, calculating M^l becomes easy. To get around this difficulty, the client can ask the server to do the computation of the Jordan form. The client only has to check that calculated value is indeed the Jordan form and that check can be done efficiently.

We have shown that it is possible to run an encrypted program on encrypted input without leaking any information about the input. Our results are the first that work for real or complex inputs whereas previous work has been confined to Z/mZ The scheme we propose introduces new methods that we believe to be of independent interest.

REFERENCES

1. Martín Abadi, Joan Feigenbaum, and Joe Kilian, "On Hiding Information from an Oracle", *Journal of Computer and System Science*, 39:1, pages 21–50, August, 1989.
2. David Aucsmith, "Tamper resistant software: An implementation", *Information Hiding - Proceedings of the First International Workshop*, LNCS no. 1174, pages 317–333, 1996.
3. D. Beaver, J. Feigenbaum, and V. Shoup, "Hiding Instances in Zero-Knowledge Proof Systems", *Advances in Cryptology – CRYPTO '90*, pages 326–338, Springer-Verlag, 1990.
4. Christian Collberg, Clark Thomborson, Douglas Low, "Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs", *Proceedings of ACM Symposium on Principles of Programming Languages*, San Diego, CA, January, 1998.
5. Christian Collberg, Clark Thomborson, Douglas Low, "Breaking Abstractions and Unstructuring Data Structures", *Proceeding of IEEE International Conference on Computer Languages, ICCL'98*, Chicago, IL, 1998.
6. Christian Collberg, Clark Thomborson, "On the Limits of Software Watermarking", *Proceedings of ACM Symposium on Principles of Programming Languages*, San Antonio, Texas, 1999.
7. Joan Feigenbaum, "Encrypting Problem Instances, or, ..., Can You Take Advantage of Someone Without Having to Trust Him", *Advances in Cryptology – CRYPTO '85*, Springer-Verlag, 1985.
8. Peter Lancaster and Mira Tismenetsky, *The theory of matrices*, Werner Rheinboldt, 1985.
9. Tomas Sander and Christian F. Tschudin, "Towards Mobile Cryptography", Technical Report, International Computer Science Institute, TR-97-049, November 1997.
10. T. Sander, A. Young, and M. Yung, "Non-Interactive CryptoComputing For NC^1". *proceedings of the 40th Symposium on Foundations of Computer Science (FOCS '99)*, pages 554-557, 1999.
11. F. B. Schneider, "Towards Fault-Tolerant and Secure Agentry", Invited paper, in *Proceedings of 11th International Workshop on Distributed Algorithms*, Saarbrücken, Germany, Septmember 1997.
12. Jan Vitek and Christian Tschudin (editors), *Mobile Object Systems: Towards the Programmable Internet*. LNCS np. 1222, Springer, 1997.