Encrypted Execution of Encrypted Programs

George Cybenko*

January 22, 2000

Abstract

In this article we show that it is mathematically possible to encrypt a program and an input dataset in such a way that the resulting encrypted objects can be executed on a general purpose computer without decrypting the objects first. The execution of the encrypted program can be monitored, traced or recorded without compromising the encryption so that no special purpose hardware is required. The encryption is as strong as possible – all programs and data sets of a fixed size are statistically indistinguishable from one another under this encryption scheme. The proposed technique uses program representations originally proposed by Sander, Tschudin and Bazzi. We extend those representations using ideas from reversible and quantum computing. The resulting program representations use unitary operators to encode the action of a program on data. We then use statistical properties of the Haar distribution over unitary matrices to show that the unitary representations can be uniformly randomized for encryption and secure execution. Techniques for efficiently constructing uniformly random unitary operators were originally developed by Stewart and are reviewed as well. The technique we propose is mathematically applicable to a large class of computations but is currently known to be feasible only for small programs on traditional computers. It appears to be feasible for all computations using quantum computers. The work described here is a promising direction for addressing problems surrounding secure execution of programs on untrusted hosts, especially involving small programs such as might arise in e-commerce.

*Author's address: Thayer School of Engineering, Dartmouth College, Hanover NH 03755, gcy@darcmouth.edu. Research partially supported by the Air Force Office of Scientific Research, the Defense Advanced Research Projects Agency, the National Science Foundation and the Department of Justice’s National Institute of Justice. All findings and opinions expressed in this article are solely those of the author.
1 Introduction

Distributed, network-based computing is rapidly growing. This has led to interest in the remote execution of programs using Java Aplets, ActiveX, JavaScript and mobile agent systems [6]. While remote execution of programs is a powerful capability, it is not without problems, specifically with respect to security.

The security threat we address in this paper has to do with the reliable execution of a program on a remote host that may or may not be “trusted.” More specifically, our goals are to send a program and dataset to a remote host, execute it there and receive the output with confidence that

- the computation was performed correctly at the remote host and;
- that the remote host learned nothing about the computation or data that was sent there.

Other security issues that arise in remote program execution, such as the important problem of protecting the remote host from malicious code, are not addressed in this paper.

Most previous approaches to this problem have been based on the technique of “code obfuscation” in which the program and data are transformed by permuting operations and using algebraic identities [4]. While code obfuscation can be effective from an engineering perspective, it has been difficult to establish any formal security properties for such approaches so far. These methods are considered ad hoc and heuristic for this reason.

In a sense, any transformation of program and data is some type of “obfuscation” but the approach we propose in this paper is qualitatively different from previous code obfuscation approaches.

In fact, our starting point is a method originally (to our knowledge) proposed by Sander and Tschudin for securely computing a function by encrypting the data, without changing the function structure itself [7]. As an example they showed how matrix multiplication could be performed by a remote host without revealing the actual matrices of interest to the end user.

Bazzi has extended this idea to encode a certain class of programs [1]. So-called “piece-wise linear programs” could be encoded in such a way that the remote host could effectively compute the required operations but the actual data, and to some extent the program, would be hidden from the
remote host. Generally speaking, the “piece-wise linear programs” studied by Bazzi are algebraically equivalent to matrix multiplication, just not explicitly represented in that way.

In this work, we introduce two ideas that extend Sander’s, Tschudin’s and Bazzi’s previous work. First of all, we use results from reversible logic and quantum computing to demonstrate that matrix multiplication is a completely general representation for a fixed step-size Turing computation. (A fixed step-size Turing computation means a fixed number of steps of a Turing computation.) Secondly, using results about the uniform distribution of unitary matrices, we show that the encryption of programs in terms of unitary matrix multiplication is effective in the sense that the probabilistic distribution of the encrypted unitary program representations is uniform – any two programs and datasets of the same size will have the same statistical distributions. This appears to be the strongest type of result possible from a theoretical standpoint.

There is a catch of course to the method we describe here. Since all programs are reduced to unitary matrix multiplication by our proposed method, the details of a program are entirely hidden in the entries of the matrix and the vector on which it operates. As is well know, when simulating a reversible or quantum computer with a classical Turing computer an exponential number of variables or states are typically required by the Turing machine. This potential exponential growth in the number of states or variables is the price we appear to have to pay for using this approach. The good news is that unitary matrix multiplication is extremely stable numerically, but this might be of little consolation if we are required to use exponentially many variables. The feasibility of this approach on traditional Turing-like machines is an area of obvious future study therefore. At the same time, we believe the basic construction is entirely feasible on a quantum computer, bearing in mind that the feasibility of quantum computing is itself an issue.

This paper draws an many complex subjects and is far from being self-contained. The reader may have to refer to the cited references for details of various constructs we describe.

Section 2 presents the basic matrix representation concept behind our approach, reviewing on the work of Sander, Tschudin and Bazzi. Section 3 deals with the relationship between unitary matrix multiplication and generalized programs using reversible logic and quantum computing ideas. Section 4 reviews properties of randomly generated unitary operators under the Haar distribution. Section 5 reviews the work of Stewart on efficiently generating random unitary operators which is necessary for implementation.
2 Encoding Matrix Multiplication

We consider the case of two networked computers, $C_l$ and $C_r$, with $C_l$ being the “local” host and $C_r$ being the “remote” host. We have in $C_l$’s memory a complex matrix, $A = (a_{ij})$ and a complex vector, $b = (b_j)$. The dimensions of $A$ and $b$ are compatible for multiplication. We wish to compute the classical matrix-vector product $c = Ab$ defined by

$$c_i = \sum_j a_{ij} b_j$$

on $C_r$ without revealing to $C_r$ what the actual matrix and vector are.

Ignoring issues of computational complexity for the moment, we can perform the following transformations on $A$ and $b$. For suitably sized matrices $U$ and $V$, we can precompute

$$A' = UAV^{-1}, \quad b' = Vb$$

and send $A'$ and $b'$ to $C_r$.

$C_r$ computes the matrix product $c' = A'b' = UAV^{-1}Vb = UAb$ and sends it back to $C_l$. At $C_l$, we compute $U^{-1}A'b' = Ab$ which is the desired result. $C_r$ saw only $A'$ and $b'$ but not $A$, $b$, or the product $Ab$.

There are some obvious questions to ask about this scheme.

- **The Complexity Question** – Why bother doing a matrix multiplication on $C_r$ if $C_l$ already has the capability?

- **The Security Question** – What can $C_r$ infer from $A'$ and $b'$?

The complexity issue has several aspects. First of all, either $A'$ or $b'$ could already be located on $C_r$ from a previous computation or could be moved there by a third party. Moreover, a sequence of operations can be performed on the encrypted data – additions, matrix-matrix multiplications and so on – at various network sites.

The problem still remains that the encoding of $A$ into $A'$ requires two matrix multiplies and the encoding of $b'$ requires one matrix-vector multiply while the actual, original computation only requires one matrix-vector computation. So we have converted an order $n^2$ complexity computation into an order $n^3$ operation, assuming the standard matrix-matrix multiplication algorithm.

The second question we need to ask is what can a remote host infer about the data and the computation. Surely, the structure of a matrix-vector product is easy to infer from the control structure of the program.
executed on the remote host, $C_r$. That is not being hidden at all. Secondly, it is not clear what invariants can be inferred about the data by $C_r$. If $U = V$, then $C' = UCU^{-1}$ will have the same eigenvalues as $C$ so this is an invariant inferrable by $C_r$, for example. If $U \neq V$, then it is less clear what can be inferred, assuming of course that $U$ and $V$ are drawn from a suitably general class of invertible matrices.

Bazzi has extended this idea to encode “piece-wise linear” programs and program segments [1]. Bazzi has shown that a large class of loop structures can be recast as matrix multiplications and so are amenable to the above encodings.

To summarize, the basic idea here is one of “data obfuscation” not “code obfuscation” since the actual data of interest to us is obscured from the remote host but the nature of the computation is not obscured in any precise formal sense.

3 Unitary Representation of Programs

The work of Sander, Tschudin and Bazzi reviewed above provides the basic idea behind an approach to encoding data for secure execution on a remote host but does not encode or hide the computation itself. The computation is matrix multiplication and that is “visible” to the remote host, $C_r$.

In this section, we develop the next ingredient of our approach by reviewing some facts from the areas of reversible computing and quantum computing. Readers are referred to [2, 3, 9] for details. The development here is very brief since the technical results are too involved to make this article self-contained.

Work by Bennett and others in the last twenty years has shown that traditional logic gates based on standard logical and, or, nor and not gates can be replaced by multiple-output gates that are reversible. These gates take three inputs and produce three outputs and can be used to implement classical logic. Since there is a one to one mapping between input bit settings and output bit settings, a reversible logic gate is merely a permutation operator on the 8-dimensional input vector corresponding to the 8 possible states of the input bits.

A permutation operator represented as a matrix is unitary. As an example consider the well-known Fredkin gate given by the following input-output table.
Fredkin gate input-output relationship [3]

This gate has three inputs and three outputs for a total of eight input and eight output states. Identifying those states with coordinates of an eight dimensional vector, the Fredkin gate can be described by the action of the permutation operator $P$ given below on an eight vector, only one of whose coordinates is 1, the others being all 0. For example, $P$ is given by the simple permutation operator

$$
P = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.
$$

Unitary operators or matrices are square matrices defined by the property that $PP^* = P^*P = I$, where $I$ is the identity. Unitary operators are invertible and their inverses are equal to their complex transpose: $P^{-1} = P^*$. Here $P^*$ is the complex transpose of $P$:

$$
P = (p_{ij}), \quad P^* = (q_{ij}), \quad q_{ij} = \bar{p}_{ji},
$$

The representation of the Fredkin gate above is one key idea behind formulating a finite sequence of any logic operations as a unitary operator. This is the basis for representing any fixed state-size Turing computation as a unitary operator, which is the state evolution operator for a corresponding
Schrodinger operator governing a quantum mechanical computing device. These observations, based on a large literature on quantum and reversible logic computing, can be summarized as follows.

**Unitary Computing Principle** – An arbitrary but finite number of steps of any Turing computation acting on an input string of data can be represented in terms of a unitary operator multiplying a vector. The unitary operator encodes the finite Turing computation and the vector encodes the input data string.

4 Randomizing Unitary Operators

The Unitary Computing Principle above tells us that any finite step-size Turing computation acting on an input data string can be cast as the product of a unitary matrix with a vector. All fixed length computations can be cast as matrix-vector products with a very special matrix structure, namely unitary. This allows us to use the constructions of Sander, Tschudin and Bazzi described in Section 2 above.

The set of unitary matrices is a noncommutative group under matrix multiplication, called the Haar measure [5]. Specifically, if \( \mathcal{U}_n \) denotes the group of unitary operators on complex \( n \)-space, \( \mathbb{C}^n \), and \( S \subset \mathcal{U}_n \) is a measurable subset of \( \mathcal{U}_n \) then the Haar probability distribution over \( \mathcal{U}_n \) satisfies:

\[
1 \geq \mu(S) = \mu(US) = \mu(SU) \geq 0
\]

for all \( U \in \mathcal{U}_n \). That is, the probability of a set is invariant under the action of multiplication by a group element. Additionally, of course, \( \mu(\mathcal{U}_n) = 1 \).

The Haar distribution is the intrinsic uniform probability distribution on \( \mathcal{U}_n \). If \( X \) is a random unitary matrix distributed according to the Haar distribution and \( U \) is a fixed unitary, then the products \( XU \) and \( UX \) are uniformly distributed according to the Haar distribution as well.

To get a feeling for this, consider the trivial case of one dimensional unitary operators, namely complex numbers of unit norm (since \( UU^* = 1 \) then means \( U \) is a complex number with \( UU = ||U||^2 = 1 \)). The group of 1-dimensional unitary operators is isomorphic to the additive group of real numbers modulo 2\( \pi \), since

\[
e^{i\theta}e^{i\phi} = e^{i(\theta + \phi)} \mod 2\pi.
\]
In this case, the Haar distribution is merely the uniform distribution over the real interval $[0, 2\pi)$.

This leads to the following result.

**Theorem** – Let $P$ be a fixed step-size Turing computation operating on a dataset, $d$. $P$ can be represented as a unitary operator $U_P \in \mathcal{U}_n$ and $d$ can be encoded as $b_d \in \mathcal{C}^n$ with $\|b_d\| = 1$ so that the result of executing $P$ on $d$ is encoded in the matrix-vector product $U_P b_d$. If $X$ and $Y$ are independent and uniformly distributed according to the Haar distribution over $\mathcal{U}_n$ then $XU_P Y^*$ is uniformly distributed in $\mathcal{U}_n$ and $Y b_d$ is uniformly distributed over the set of vectors in $\mathcal{C}^n$ of length one. Moreover, the objects thus encrypted are statistically independent of one another.

This result states that if we encode two programs, $P_1$ and $P_2$ as $U_1, U_2 \in \mathcal{U}_n$ and encode two input strings, $d_1$ and $d_2$ as $b_1, b_2 \in \mathcal{C}^n$, generate independent $X_1, X_2, Y_1, Y_2 \in \mathcal{U}_n$, then the program representations $X_1 U_1 Y_1^*$ and $X_2 U_2 Y_2^*$ will be independently distributed according to the same Haar distribution. Moreover, $Y_1 b_1$ and $Y_2 b_2$ will also be independently, identically distributed, uniformly over the unit sphere in $\mathcal{C}^n$ as well.

5 Efficient Generation of Random Unitary Operators

The key to the encryption concept identified above is the effective generation of unitary matrices, distributed according to the Haar probability distribution on $\mathcal{U}_n$.

G.W. Stewart discovered an extremely elegant and simple algorithm for generating such random unitary matrices [8]. We review that work briefly.

Start by generating a complex $n$ by $n$ matrix, $W$, each of whose entries is distributed according to a scalar, complex normal distribution with zero mean and common fixed variance, $\sigma^2$. The complex QR factorization of $W$, namely $W = QR$, is a unique factorization of $W$ into a product of a unitary $Q$ and an upper triangular matrix $R$ with nonnegative diagonal entries.

Stewart observed the following: if we pick any fixed $X \in \mathcal{U}_n$, then $XW$ has entries distributed the same as the entries of $W$. Therefore the distribution of $Q_X$ in the QR factorization of $XW = Q_X R_X$ is the same as the distribution of $Q$ in the factorization of $W = QR$. Therefore, the $Q$ factor of $W$ generated as above is distributed according to the Haar distribution.
The algorithm for computing such a random unitary is quite simple then:

1. Generate a square matrix, $W$, whose entries are complex numbers, normally distributed with zero mean and common variance, $\sigma^2$;

2. Compute the complex QR factorization of $W$.

The resulting $Q$ factor will be uniformly distributed in $\mathcal{U}_n$ according to the Haar distribution.

6 Summary and Discussion

The technique we describe here is formal, not practical, for encrypting a fixed number of steps of a Turing computation and a corresponding input datastring.

The steps are as follows:

1. Encode a Turing computation with a fixed number of steps as a reversible or quantum computation, resulting in a representation of the program as a unitary matrix operator. Encode the input datastring as a vector, $b$, in a corresponding manner. The resulting program encoding is a unitary matrix, $U \in \mathcal{U}_n$ for some $n$. The resulting output string is encoded by the product $Ub$.

2. Generate two independent identically distributed unitary matrices from the Haar distribution over $\mathcal{U}_n$, say $X$ and $Y$.

3. Precompute $U' = XUY^*$ and $b' = Yb$ locally and send these to the remote host, $C_r$.

4. $C_r$ computes the product $c' = A'b' = XUY^*Yb = XUb$ and send this result back to $C_l$.

5. $C_l$ computes $X^*c' = X^*XUb = Ub$ which is the encoded, desired output of the original program.

We can tell if a computation has been tampered with by the remote host by embedding simple checksums or even constants into the encrypted programs. Since the randomization affects all entries of the data and the program uniformly, the probability of a random change leading to the correct resulting checksum or constant is basically equivalent to randomly
guessing the correct answer to the program. We have not developed this idea in a quantitative way yet.

This encryption scheme is not presently practical for classical Turing computations on classical Turing machines. It does however appear to be practical on a quantum computer, if quantum computers themselves are practical.

A randomly distributed unitary can be described by a single number, namely the seed used to generate $n^2$ random, normal complex numbers. Thus the "key" for a random unitary is easy to communicate.

This approach does not make sense from a complexity perspective if all the encryption is done on the local host, $C_i$. It does however make sense when data and programs are distributed or when the same program will be executed on many datastrings remotely. Then the $n^3$ costs of matrix multiplication for encryption can be amortized over many remote executions of complexity $n^2$ to perform the remote matrix-vector multiply in encrypted form.

Much work remains to be done on the efficiency of this approach for traditional Turing machines. Bazzi has shown that the Sander/Tschuldin basic idea can be extended to cover a larger class of programs, so-called "piece-wise" linear programs. In this paper, we have shown that the basic concept can be extended to all fixed step-size Turing computations at the expense of possible exponential growth in representation size. This leaves the middle ground open – namely, for what classes of programs between the class of piece-wise linear programs and fixed step-size Turing computations can this method be made practical?

References


