Preserve Privacy Data Using Stile Methods

P.Nithya¹ and M.Lakshmi prabha²

Student, Gnanamani College of Technology, Namakkal, Tamil Nadu, India¹
Assosiate Professor, Department of CSE, Gnanamani College of Technology, Namakkal, Tamil Nadu, India²

Abstract-Preserve privacy data using sTile methods addresses the challenge of executing computations on untrusted machines in a trustworthy manner. Its focus is on preserving data privacy while solving computationally intensive problems on untrusted machines. Existing work presented sTile technique for building software systems that distribute large computations onto the cloud while providing guarantees that the cloud nodes cannot learn the computation’s private data. sTile is based on a nature-inspired, theoretical model of self-assembly. In this paper, we present a prototype implementation that solves NP-complete problems. sTile explores the fundamental cost of privacy through data distribution. Tile assemblies are Turning universal, so future extensions of sTile can be made to perform arbitrary computations and to automatically compile programs into tile assemblies.

Index terms- Self-assembly, Adder, Multiplier; sTile, tile assembly model

I. INTRODUCTION

The emergence of cloud computing is evolving the nature of computation. Instead of using private machines, users allow the cloud to maintain, manipulate, and safeguard their data. This evolution has allowed ubiquitous access to computation and data with higher availability and reliability than possible with personal machines and local servers.

This paper addresses the challenge of executing computation on untrusted machines in a trustworthy manner. Its focus is on preserving data privacy while solving computationally intensive problems on untrusted machines. We present sTile, a technique for building software systems that distribute large computations onto the cloud while providing guarantees that the cloud nodes cannot learn the computation’s private data. sTile is based on a nature-inspired, theoretical model of self-assembly. While sTile’s computational model is Turing universal, in this paper, we present a prototype implementation that solves NP-complete problems. sTile explores the fundamental cost of privacy through data distribution.

Self-assembly, the process by which objects autonomously come together to form complex structures, is omnipresent in the physical world. A systematic study of self-assembly as a mathematical process has been initiated. The individual components are modelled as square tiles on the infinite two-dimensional plane. Each side of a tile is covered by a specific “glue”, and two adjacent tiles will stick if they have matching glues on their abutting edges.

sTile computes while preserving privacy by breaking a computation into small pieces and distributing those pieces onto a large network. Each piece is so small that it is prohibitively difficult for an adversary to collect enough pieces to reconstruct the confidential data.

Existing approaches to using the Internet’s computational resources either assume reliable and trustworthy underlying machines only store data privately and rely on trusted entities to compute or are theoretical constructs (e.g., quantum computing and homomorphic encryption) that, to date, have not produced implementations efficient enough for practical use.

We evaluate sTile in three ways:

1. First, we formally prove that sTile systems preserve data privacy as long as no adversary controls more than one-half of the cloud.

2. Second, we empirically demonstrate sTile’s feasibility by deploying an open-source, publicly available prototype implementation on three distinct networks, including the globally distributed PlanetLab testbed.
3. Third, we formally analyze the communication and computation costs induced by sTile, bound them, and empirically verify those bounds. sTile significantly outperforms existing cryptography-based privacy techniques, such as homomorphic encryption.

2. CONCEPTS

2.1 Addition
sTile preserves privacy by breaking a computation into small pieces and distributing those pieces onto a large network. Each piece is so small that it is prohibitively difficult for an adversary to collect enough pieces to reconstruct the confidential data.

In this section, we describe sTile with an example of distributing an addition computation. To describe adding using sTile, we explain three separate elements of our solution:

i) The addition tile assembly
ii) The distribution process
iii) The source of privacy

i) The Addition Tile Assembly
A tile assembly is a theoretical construct, similar to cellular automata. It consists of square tiles with static labels on their four sides. Tiles can attach to one another or to a growing crystal of other tiles when sufficiently many of their sides match. Fig. 1a shows eight different types of tiles used for addition. These tile types are the program—the tile assembly encoding of the algorithm for adding two integers, in binary, one bit at a time. Fig. 1b shows a seed crystal that encodes an input: 10 (1010 in binary) in the top row and 11 (1011 in binary) in the bottom row. When an instance of a tile from Fig. 1a matches the seed crystal on three sides, that tile instance attaches to the crystal. Fig. 1c shows the seed with a single attached tile. Note that this tile adds the least significant bit of each input: 0 Æ 1 ¼ 1, displayed in the center of the newly attached tile. The label on the west side of the newly attached tile is the carry bit: 0. The tiles execute full adder logic to add the bits, one at a time, eventually producing the sum 21 ¼ 101012 in the middle row of Fig. 1d.

ii) The Distribution Process
sTile uses the theoretical tile assembly to decompose a computation into small parts. Each small part represents a tile. Fig. 1e shows how the 10 Æ 11 execution might be deployed on eight network nodes. Each node only deploys tiles of a single type, designated by the client machines. The client sets up a seed on
thenetwork by asking nodes that can deploy tiles of appropriate types to deploy instances of the deploying and maintains references to the geometrically adjacent tile instances on other nodes. Next, tiles with an empty adjacent location coordinate with their crystal neighbors to recruit matching tiles to attach.

Once the execution finishes, the tiles in the middle row report the solution to the client, indicating the node IDs of their crystal neighbors, which the client uses to reconstruct the output.

iii) The Source of Privacy

Each tile instance is aware of only a single bit of the input, output, or intracomputation data, and not of the bit’s global location. An adversary may attempt to reconstruct the confidential data from the nodes it controls. For example, Fig. 1f shows an adversary that has compromised three nodes (2, 3, and 6), and now has access to the data in five tiles. However, this adversary can only tell that there are some 0 and 1 bits scattered throughout the input, the computation, and the output, but not how many and not their relative positions. In fact, in this example, no three nodes contain the entire input (nodes 1, 2, 5, and 7 deploy the input).

2.2 sTile

sTile is a technique for designing, implementing, and deploying software systems that distribute computation onto large, insecure, public networks. sTile’s primary concern is to perform computation while preserving the privacy of the involved data. Figure 2 shows a high-level overview of sTile.

sTile consists of four components:
1. The tile assembly and the associated algorithms
2. The corresponding tile architecture and the associated algorithms
3. The Mahjong implementation framework

2.2.1 Computing with Tiles

A key component of sTile is a tile assembly. Tile assemblies are theoretical objects that have no notion of privacy, although it is their basic structure that allows sTile to preserve privacy. sTile is a reification of a tile assembly as a distributed software system.

Figure 3 shows the 64 possible types of tiles of the 3-SAT solving assembly. The tiles “communicate” via their side interfaces. Some interfaces contain a 0 or a 1, communicating a single bit to their crystal neighbors. Other interfaces include special symbols such as v and :v indicating that a variable is being addressed. ? meaning that a comparison should take place, ? meaning the given tile attaches nondeterministically, and j and k indicating the correctness of the computation up to this point.

2.2.2 Tile Architecture and Algorithms

AsTile-based system is a software system that uses a network of computers to solve a computational problem. Intuitively, the network simulates a tile assembly: Each computer pretends to be a tile or many tiles, and communicates with other computers to self-assemble a solution to a computational problem. Each computer deploys tile components, each representing a tile in a tile assembly, and facilitates the proper communication channels and algorithms to allow the tile component self-assembly. Thus, a tile architecture is based on a tile assembly; the software system employing that architecture solves the particular computational problem that the tile assembly solves.
2.2.3 Mahjong implementation framework.

The final element of sTile is the Mahjong implementation framework which uses the tile architecture and algorithms to automatically compose a sTile-based software system on a network. The open-source Mahjong framework is realized as a Java-based middleware platform that faithfully implements the tile architecture and its algorithms.

Mahjong’s implementation uses Prism-MW, amiddleware platform that provides explicit implementation-level constructs for declaring components, interfaces, interactions, network communication.

3. COMPUTATIONAL FEASIBILITY

To demonstrate that sTile is a feasible solution for building software systems that distribute computationally intensive problems on very large networks, we must show that
1) such systems’ computational speed is proportional to the size of the underlying network,
2) such systems are robust to network delay,
3) real-world-sized problems can be solved on real-world-sized networks in reasonable time.

3.1 sTile-Based Implementations

Simjong is a Java-based discrete-event simulator with network-delay simulation capabilities. Simjong executes on a single machine and creates a user-specified number of virtual hardware Node components, each capable of deploying tiles. A central Clock component keeps track of virtual time and allows each Node to execute one instruction per clock cycle.

While executing, Simjong keeps track of the number of completed seeds and reports its progress. Thus, it is possible to use Simjong to estimate the time required for a computation to complete after executing only a fraction of that computation.

3.2 Experimental Setup

We use three distributed networks for our experimental evaluation:
1) a private heterogeneous cluster of 11 Pentium 41.5-GHz nodes with 512 MB of RAM, running Windows XP or 2000;
2) a 186-node subset of USC’s Pentium 4 Xeon 3-GHz High Performance Computing and Communications cluster
3) a 100-node subset of PlanetLab, a globally distributed network of machines of varying speeds and resources that were often heavily loaded by several experiments at a time.

The cross section of data we present in this paper used four representative instances of NP-complete problems, to which we will refer by their labels:
- A: 5-number 21-bit SubsetSum problem
- B: 20-variable 20-clause 3-SAT problem
- C: 11-number 28-bit SubsetSum problem
- D: 33-variable 100-clause 3-SAT problem.

Our experimental goals were to verify sTile’s scalability with respect to network size and robustness to network delay. Our experiments had three independent variable:
1. The number of nodes,
2. The network communication speed between nodes
3. The size of the NP-complete problem—and one dependent variable—the time the computation took to complete.

First, to verify the correctness of sTile-based systems, we used Mahjong-based implementations to solve over one hundred of SubsetSum and 3-SAT problems, including A, B, and C. As a rule of thumb, we chose the sizes of problem instances to each execute in under 4 hours on our 186-node cluster. We verified that, on each of the above three networks, the implementations found the correct solution to each instance, it sent no unexpected communication between network nodes and no node produced undesired connections between tile components. Further, we verified that, when provided inputs with a negative answer, the implementations continued to execute indefinitely, as expected.

3.3 Scalability

To verify that the speed of the computation is proportional to the number of nodes on the underlying network, for each of the three networks described above, we deployed Mahjong-based implementations on the entire network and on randomly selected halves of the network. We varied the size of the problem and
measured the average time in which the implementations found the solution over 20 executions. We then also deployed Simjong on virtual networks of increasing size from 125,000 to 1,000,000 nodes. This allowed each Simjong execution to complete in about an hour of actual time, while executing a sufficiently large number of seeds. Our measurements have shown that, after the first few thousand seeds, our implementations make fairly constant progress through the seeds and that extrapolating from the 10.4 percent fraction is accurate.

3.4 Robustness to Network Latency

Intuitively, high-network latency should adversely affect the speed of sTile-based systems: If the tile attachments happen sequentially, the latency affects every attachment and greatly slows down the overall computation. However, in the case of NP-complete computations, this intuition is false. In such computations, many of the tile attachments happen independently, in parallel. Each node in our experiments deployed millions of lightweight tiles, and whenever a sTile packet traveled between nodes, those nodes handled other tiles rather than waiting idly for the network communication to arrive. As a result, the throughput of sTile-based systems is not affected by the network latency.

4. PRIVACY PRESERVATION

sTile’s privacy preservation comes from each tile being exposed only to a few intermediate bits of the computation and the tiles’ lack of awareness of their global position. To learn meaningful portions of the data, an adversary needs to control multiple, adjacent tiles. We call a distributed software system privacy preserving if, with high probability, a randomly chosen group of nodes smaller than half of the network cannot discover the entire input to the computational problem the system is solving. We argue that neither

- i) a node deploying a single tile, nor
- ii) a node deploying multiple tiles can know virtually any information about the input; moreover,
- iii) controlling enough computers to learn the entire input is prohibitively hard on large networks.

1. Each tile type in an assembly encodes at most one bit of the input. A special tile encodes the solution, but has no knowledge of the input. A node that deploys a single tile is only able to learn information such as “there is at least one 0 bit in the input,” which is less than one bit of information.

2. Each node on the network may deploy several tiles. However, each tile is only aware of crystallike neighboring tiles and not of its global position. Thus, a node deploying several noncrystal-like neighboring tiles cannot reconstruct anymore information than if it only deployed a single tile. The only way the node may gain more information is if it deploys crystal-like neighboring tiles.

3. Suppose an adversary controls a subset of the network nodes and can see all the information available to each of the tiles deployed on those nodes. Then, the adversary can attempt to reconstruct the computation’s input from parts of the crystal that consist of tiles deployed on the compromised nodes.

5. RELATED WORK

Distributing computation: The growth of the Internet has made it possible to use public computers to distribute computation to willing hosts. This notion focuses the underpinning of computational grid. Among systems that concentrate on distributed computation are BOINC systems (such as SETI@home and Folding@home), MapReduce, and the organic grid. A unique approach—FoldIT—uses the competitive human nature to solve the protein-folding problem. These systems try to solve exactly the highly parallelizable problems toward which our work is geared, but unlike sTile, they do not preserve privacy.

Cloud privacy: Cloud computing has reemphasized the importance of data privacy, causing the emergence of numerous approaches for keeping data private on the cloud. Most such approaches concentrate on private data storage and user-authorized data retrieval and require some trusted agents whereas our work concentrates on preserving privacy during computation and requires no trusted agents.

Privacy-preserving computation: In classical computing, it is not possible to get help from a single entity in solving an NP-complete problem without disclosing most of the information about the input and the problem one is trying to solve [19]. Our approach avoids this shortcoming by distributing such a
request over many machines without disclosing the entire problem to any small enough subset of them.

6. CONTRIBUTIONS

sTile distributes computation onto large, insecure, public networks in a manner that ensures privacy preservation, fault and adversary tolerance, and scalability. We presented a rigorous theoretical analysis of sTile and formally proved that the resulting systems are efficient and scalable, and that they preserve privacy as long as no adversary controls half of the public network.

We deployed two sTile implementations on several networks, including the globally distributed PlanetLab, to empirically verify

1. The correctness of sTile algorithms,
2. The speed of sTile computation is proportional to the number of nodes,
3. Network delay has a negligible effect on the speed of the computation, and
4. Mathematical analysis of the time needed to solve large problems on large networks is accurate. For networks larger than about 4,000 nodes, sTile outperforms optimized solutions that assume privately owned, secure hardware.

sTile explores the fundamental cost of achieving privacy through data distribution and bounds the extent to which a privacy-preserving system is less efficient than a non-private one. While that cost is not trivial, we have demonstrated that sTile-based systems execute orders of magnitude faster than homomorphic encryption systems, the alternative promising approach to preserving privacy.

7. CONCLUSION

In this paper we conclude that the proposed work demonstrated how sTile can solve 3-SAT (Satisfiability). sTile solves all NP-complete problems without designing new assemblies. NP-complete computation can be accelerated by developing faster algorithms for single machines and small clusters. Such work is complemented to ours since sTile is based on a Turing-universal computational model and can implement each of these advanced algorithms on large distributed networks.