Using a Network of Untrusted Computers for Secure Computing

Michal Malý
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics, Comenius University
Bratislava, Slovak Republic
Email: maly@ii.fmph.uniba.sk

***Abstract*—This paper defines a new problem in distributed computing: How to securely compute one's own problem using a network of untrusted computers? A theoretic solution of the problem is also presented. All secret input data and a secret result are known only by the initiator of the computation. Other computers are used only to carry out the computation using their computer time and memory. The communication can be eavesdropped, and any of untrusted computers can arbitrarily tamper the computation, assuming that no cooperation between untrusted computers occurs. This problem is different from a well-known multiparty distributed computation problem. Here, only one party has all the secret data. The computation is carried out on untrusted computers. An untrusted computer has access to only a small part of secret. Provided no or only a few untrusted computers are cooperating, the secret – input data, and the result of the computation – is not revealed.**

Keywords*- *cloud computing; security; distributed computing.

I. Introduction

Suppose a company wants to solve a confidential computational task, but they do not own enough computers to carry out the computation on their own computers. They could rent some computer time from another company to carry out the computation on their computers, but they are afraid that the company would misuse the confidential data and the result of the computation.

However, they can rent some computer time from many companies. They believe that those concurring companies would not cooperate together. The computation could be distributed among many untrusted units, assuming that the attack is not synchronized among multiple untrusted parties.

The problem is to execute the computation on untrusted computers so each of the computers has access to only a small part of the computation, i.e., either only to every j-th tape position or to every j-th written symbol for a chosen j. This means, if a confidential document is stored on the tape, an attacker can read it only discontinuously, e.g., only one bit from every ASCII character. This suffices to keep the content confidential. Current methods in distributed computing do not provide any guaranteed protection of this form.

The structure of the paper is as follows. We investigate alternatives which provide at most practical difficulty or provide only protection of stored data. Also related problems in multi-party computation and mobile cryptography are presented. A framework of computers simulating a Turing machine by message passing is presented, first dealing with passive interception and then handling active messages tampering.

II. Alternatives

A. Hiding computation problem

The company could attempt to hide the computation in some way of obfuscating it enough, to prevent anyone from reading the result easily. The company can for instance run a virtual machine inside the provider's computer. If virtualization is done by proprietary software, it can be challenging to analyze all data structures by the potential attacker to figure out what computation is done inside the virtual machine. However, this is only "obfuscation" and should not be considered to be a good practice. A close analysis or some background info can in principle remove the obfuscation layer, uncovering the secrets. The protection should come from a logical impossibility rather than practical or technical difficulty.

B. Hiding data problem

On the other side, if the company's problem would be only storing a vast amount of data, they could simply rent some storage and keep their data in encrypted form there. As long as the act of decrypting is carried on trusted computers, no information leak can occur.

III. Related work

A. Use of distractive units

A method for distributed computation using distractive computational units was patented by Google [2]. The computation is partitioned into computational units and a number of distractive units is generated. All units are forwarded to providers. The results are collected and evaluated in order to obtain the final result. It is supposed, that the distractive units can substantially inhibit the reconstruction of the secret data.

ISBN: 978-1-61208-006-2

B. Secure multi-party computation

A related important problem in cryptography, first introduced in [13], is to enable two or more parties to publicly compute a shared function from secret inputs provided by the parties, guarantying the secrecy of inputs and the correctness of the result. The notorious example are two millionaires, who wish to know, which of them is richer, without revealing the exact wealth. Effective protocols providing protection against active adversaries were proposed by [4] [5] [7] and for multiple parties assuming some level of honesty in [1].

C. Volunteer computing

Computer owners readily donate their computer time to interesting projects, such as prime search [6], search for extraterrestrial intelligence [11], and simulations of protein folding [12]. These projects require dealing with an occasional volunteer computers malfunction, intentional fraudulence in order to gain extra credit, or sabotage. The techniques include majority voting and more advanced "spot-checking" [9].

In volunteer computing, each participant has access to the data, which he obtained, and partial result of the computation, which was carried out on his computer. In "public" projects, there is no need to keep it in secret. On the contrary, the result is often shown to the user in order to attract more volunteers, or even a prize is awarded for successful computation.

However, in such conditions, certainly no military computing project could be carried out on volunteer computers.

D. Mobile cryptography

In [8], a homomorphic encryption scheme was proposed to enable secure computation on a mobile agent residing on a potentially untrusted hardware. Unfortunately, only encrypted computation of polynomial functions is known.

IV. Problem specification

Let us suppose the company can gain (rent) any reasonable number of computers, each from another party (e.g., in a volunteer project). Let us suppose the parties are not cooperating: Although each untrusted computer is under somebody else's control, nobody has control over two different computers. The goal is to execute a computation on private data, so only the originator of the computation has access to data and to the result of the computation. Each untrusted computer will have access only to a arbitrarily small part of the secret.

A. Premises

The network is considered fast enough and the number of network messages used to accomplish the computation is allowed to be linear with the time of computation.

V. Method

The possibility of such computation will be demonstrated using a classical Turing machine model to formalize the computation. The process of computation will rely on standard cryptography mechanisms:

- To ensure confidentiality of transferred messages, the originator generates a digital signature for himself and for each host to encrypt and sign messages.
- To prevent tracing of messages, the mechanism of anonymous routing called "onion routing" [3] and implemented in TOR [10] is used. The originator and each hosts connects to the TOR network, and only originator knows the role of each host.

The process will be accomplished in two steps. First, I will suppose nobody is actively tampering the computation, only passive eavesdropping is allowed. Assuming this I prove that only an arbitrary small part of computation is revealed to each untrusted side.

The second step involves the mechanism of majority voting, which at each step of the computation, that a malicious message will be detected and the attacker will be revealed.

VI. Description

The framework uses a network of computers passing messages between them. The network as a whole will do the computation. To make the computation secret, the computation is split to small parts. No computer in the network can gain a large amount of information at one time. The order of computers participating in the computation is devised so as no host can intercept successive parts of the computation.

To describe the computation, we will use the Turing machine. Every movement of the head will be a small, atomic part of the computation. One part consists from messages which are passed between respective computers. After executing this part, the computation moves one step ahead, and another sequence of messages is passed between another set of computers.

Theorem 1: Let us have a Turing machine $A = (K, \Sigma, \Gamma, \delta, q_0, F)$, where $|K| = k$, and t be the time, i.e., the t-th step of the computation. Let $n > 1$ be a natural number. Let w be the input word. Then it is possible to execute the computation of the machine A on the word w using a network of $k{*}n{+}n$ independent untrusted computers, so that if nobody is tampering the computation:

1) if $A(w)$ stops after T steps, then the execution in the networks stops after T iterations, gives the same result, and the number of messages in the network is $5 * T + c * k * n$ where c is some constant.
2) i-th computer can only have access to $(n * j + i)$-th position on the tape during the whole computation for any j and $0 \leq i < n$ and $(n + r * k + s)$-th computer can only have access to the symbols read or written to

the tape in $(n * j + r)$-th step of network computation where $0 \leq r < n$, $0 \leq s < k$ and any j.

Proof: The setup is as follows: The host computers are split into two groups. First group of n computers will represent the tape; second group of $k * n$ computers represents the head of the machine. Computers in first group will form a cycle, so i-th computer represents every $(n * j + i)$-th position on the tape for any j and for $0 \leq i < n$.

Computers in the second group will represent the head, i.e., the finite-state automaton. To accomplish this, let us take k computers, associate each of k states of the automaton to one of k computers, and program the computer so it knows the values of delta function for its associated state. Now we have this distributed automaton, which is able to act as a head of the machine. But if the delta function is constructed so the automaton remains in one state for a long time, one computer would be able to intercept a long part of the computation.

To avoid this, we make n copies of this automaton, and rewire the outputs of delta function to point to the respective state, but not in the same copy as the current computer (state), but in the following copy of the automaton. The computers in the last copy of the automaton will point to the computers in the first copy, so the copies form a cycle (in some sense). Now we connect the head to the tape. We describe one transaction, which represents one movement of the head. The transaction consists of the following messages (see Fig. 1):

1) The originator obtains addresses of the network computers.
2) The originator generates number identification for each computer (according to the numbering in this proof).
3) The originator generates digital keys for each of computers and uploads them.
4) Each of the computers joins TOR network and communicate its address within TOR network to the originator.
5) The originator discloses number identification, public key, and TOR address of each computer to every computer.
6) The originator uploads an computing protocol application on each computer.
7) The originator fills out input data on tape.
8) The originator starts the execution.
9) The network computation iterates, each iteration is composed of an atomic transaction between 3 computers, see below.
10) When the computation stops in one of final states, the computer representing the state sends message to the initiator.
11) The initiator is now able to retrieve the content on the tape.

The transaction (see Figure 1) runs between 3 computers: The computer representing the current state of the automaton, the computer representing current tape position and the computer representing the new tape position.

Suppose we are in the t-th step of computation and Turing machine state is the s-th state ($0 \leq s < k$) and tape is in position p. Let r, r' be remainders of $t, t+1$ divided by n.

First, the current-state computer ($(n + r * k + s)$-th) asks for the symbol on the tape. It receives an answer from the current-tape-position computer ($p \text{ modulo } n$). Let this symbol be x. According to the delta function it decides what is the new state (s'), what symbol is to be written on the tape (y), and where to move the head (direction $d \in \{-1, 0, 1\}$), i.e., $\delta(q_s, x) = (q_{s'}, y, d)$. Therefore it sends a message with new symbol y and the movement direction d to the current-tape-position computer, which will remember the new symbol, and will send a message to the computer which represents the new tape position ($(p + d) \text{ modulo } n$). This message will authorize the current state to announce the new state ($(n + r' * k + s')$-th computer) to the new-tape-position computer. Which computer will be the new-state computer is derived from the delta function and from the number of copy of the automaton, in which belongs the current-state computer. The new-state computer is chosen to represent the new state s', but in the next copy r' of the automaton. This new-state computer is thus authorized to read the symbol from the new position. ■

VII. Byzantine computing

How can we assure nobody is actively modifying the computation? We could use more computers, to check them each other. In principle we run the same computation on more computers and detect if the outcome (the messages sent) differs (majority voting).

Suppose at most one computer is violating the protocol, and we want to detect this violation and report the violator. If more computers will be cheating, the result is undefined.

The setup is as follows. Every computer is tripled (create a copy of it, assign the copies the same role, only the public key and TOR address is different). Messages previously sent to the computer are now sent parallel to each of its copies. Now each computer waits until it receives messages from all copies of its predecessors. It compares the received messages. If they are different, it reports to the originator and sends received messages. Messages are signed, so the report cannot be faked. According to this report from all three copies, the central computer can determine who violated the protocol.

A timeout can be defined to enable detection of sabotage by not sending messages. After receiving the first message, the computer waits until all messages arrive. A missing message is reported after the time-out to the originator.

The number of computers allowed to violate the protocol (v) can be extended in a similar manner using $2v + 1$

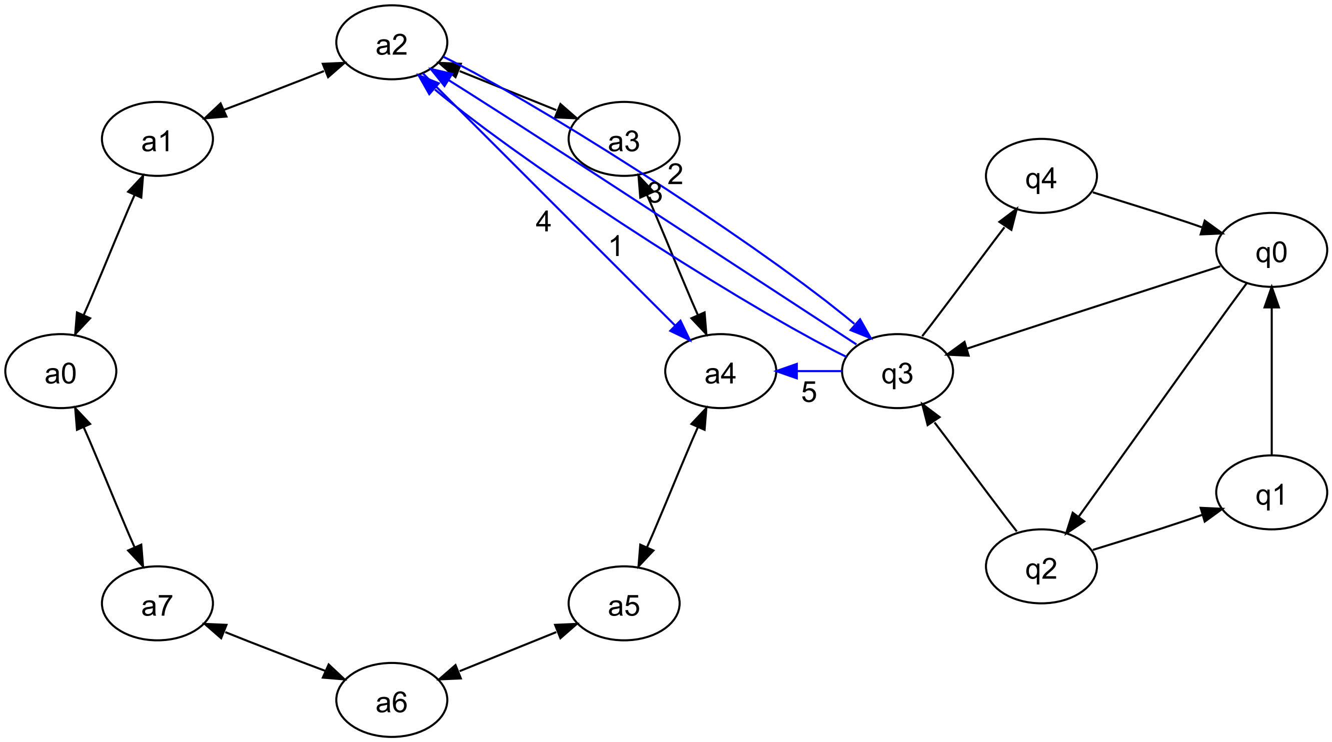

Figure 1. Example of an atomic transaction. The states a_y represent the tape, the states q_x represent the automaton. For simplicity we show only one copy of automaton. Black arrows are the possible transitions between states, blue arrows are messages. Messages: #1 What is on the tape? #2 Symbol 'x'. #3 Write 'y' and move head left. #4 we are moving left and q3 is the old state. #5 The new state is q4 (but from the next copy of automaton).

independent copies.

VIII. Applications

Perpetually growing business of computer outsourcing or modern trends such as web applications can release us from owning and maintaining hardware. The presented method allows us to not resign on security issues. The method can be used even in intra-company applications: The company could have a number of not-so-trusted administrators. The system based on this computational model could save the company from the possibility of disclosing the secrets to them. A military, or other confidential research could be realized on a number of volunteer computers.

IX. Conclusion and Future Work

We have presented a method to run a computation on a number of computers, where no computer has a complete access to the computation and the result of the computation is not revealed. In fact, the part of the computation revealed to a single computer can be made arbitrarily small.

The classical Turing machine model is not flexible enough to reflect current hardware possibilities, and was used to demonstrate the theoretical possibility of the problem solution. Another, more refined model based on RAM (random access machine), using individual computers as separate registers, will be investigated in future work.

Even using the Turing machine model, some bad-designed programs can reveal the whole secret, when the head is accessing the tape too many times. This can be possibly solved by re-designing the program. The number of messages is huge, and similarly the number of used computers is relatively high. However, the method demonstrates the theoretical possibility of trusted computing on untrusted computers.

References

[1] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 11–19, New York, NY, USA, 1988. ACM.

[2] S.N. Gerard. Distributed computation in untrusted computing environments using distractive computational units, February 9 2010. US Patent 7,661,137.

[3] D. Goldschlag, M. Reed, and P. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150. Springer, 1996.

[4] Stanisaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 97–114. Springer Berlin / Heidelberg, 2007.

[5] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer Berlin / Heidelberg, 2007.

[6] Mersenne Research, Inc. Great Internet Mersenne Prime Search. Available at: http://www.mersenne.org, 1996–2011. Last access: May-11-2011.

[7] Jesper Nielsen and Claudio Orlandi. Lego for two-party secure computation. In Omer Reingold, editor, *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer Berlin / Heidelberg, 2009.

[8] T. Sander and C. Tschudin. Protecting mobile agents against malicious hosts. *Mobile agents and security*, pages 44–60, 1998.

[9] L.F.G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems* 1. *Future Generation Computer Systems*, 18(4):561–572, 2002.

[10] The Tor Project, Inc. Tor: anonymity online. Available at: https://www.torproject.org, 2002–2011. Last access: May-11-2011.

[11] University of California. Seti@home. Available at: http://setiathome.berkeley.edu, 2011. Last access: May-11-2011.

[12] Vijay Pande and Stanford University. Folding@home. Available at: http://folding.stanford.edu/, 2000–2011. Last access: May-11-2011.

[13] Andrew C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, pages 160 –164, nov. 1982.