Poster: Garbled Computations: Hiding Software, Data, and all Computed Values

Omar Abou Selo*, Chris Clifton†, Mikhail J. Atallah†, Qutaibah M. Malluhi*, Abdullatif Shikfa* and Yongge Wang‡
*Qatar University
{omaraselo, qmalluhi, ashikfa}@qu.edu.qa
†Department of Computer Science, Purdue University, West Lafayette, IN
matallah@purdue.edu, clifton@cs.purdue.edu
‡Department of SIS, UNC Charlotte, NC, USA
yongge.wang@uncc.edu

***Abstract*—Following is a new protocol for private function evaluation using 3 non-colluding parties: Alice, Bob, and Carol. A program owner and a data owner perform an initial setup with Alice, Bob, and Carol and can then go offline until the computation is done and the result is returned to either of them. Throughout the execution, the program and data remain hidden and do not leak any information to Alice, Bob, Carol or the original owners.**

I. Introduction

There have been substantial developments in computing on encrypted data without revealing the data contents. Tools such as Secure Multiparty Computation (SMC), Homomorphic Encryption, and Oblivious Computation have made great progress toward this goal.

On the other hand private function evaluation (PFE), which is the problem of hiding both the program and the data did not see as much development. There are solutions that depend on running universal circuits in SMC protocols [1] making the program part of the input therefore hiding the program too, but these incur additional logarithmic overhead to the complexity and are difficult to implement. More recent solutions do not use universal circuits [2] thereby avoiding the logarithmic overhead complexity but are still not practical.

This work introduces a new three-party protocol that attempts to make practical PFE possible. The protocol uses *One Instruction Set Computer* (OISC) in combination with SMC and additive homomorphic encryption (specifically Paillier [3]) to hide the instructions' execution. The following section provides a brief high-level description of the protocol.

II. Protocol

A. OISC

Using a OISC hides the nature of the instruction being executed. Since there is only one type of instruction to execute, knowing the instruction will not leak information about the program. While there are several possible Turing-complete instructions that can be used for OSIC, SUBLEQ stands out as it provides good functionality and already has a compiler from a C like higher language [4].

Assuming a SUBLEQ program $\mathcal{P}$ and its input data $\mathcal{D}$, a SUBLEQ instruction in this protocol has four memory address operands A, B, C and D that performs the following:

1) $\mathcal{D}[B] = \mathcal{D}[B] - \mathcal{D}[A]$
2) $if(\mathcal{D}[B] \leq 0)$ go to instruction $\mathcal{P}$[C], otherwise to $\mathcal{P}$[D]

Note that this SUBLEQ instruction has two differences from the generally used one. First, the instruction has one additional operand D, such that when if condition is false $\mathcal{P}$[D] is executed instead of the very next instruction in the program. Second, the program and data are separated such that A and B address the data memory while C and D address the program memory.

B. Initial phase

There are three computing facilities (e.g., cloud service providers): Alice, Bob, and Carol. In addition, there is a program owner who owns a SUBLEQ program $\mathcal{P}$ and there is a data owner who owns a data set $\mathcal{D}$. Assume that the program $\mathcal{P}$ runs on the data set $\mathcal{D}$ for a maximum of T rounds (A round represents the execution of one SUBLEQ instruction). Carol sets up T Paillier encryption schemes and publishes the T round public keys for encrypting the program sequence $\mathcal{P}$ for each round. He also publishes one additional key for encrypting $\mathcal{D}$.

The data owner generates a random permutation π_d to permute and encrypt his data using the session data encryption key. The data owner sends her permutation function π_d to the program owner and sends $E(\mathcal{D}_{\pi_d})$ to Alice. The program owner generates a random permutation π_p to permute and encrypt his program using the round 0 program encryption key, and computes the current instruction pointer $i_c = \pi_p(0)$. The program owner sends $E(\mathcal{P}_{\pi_p})$ together with the current instruction pointer i_c to Alice.

C. High level round phase

In a high level view, the goal of each round is for Carol and a second party member (either Alice or Bob) to execute one instruction then re-permute and re-encrypt the program and

This publication was made possible by the NPRP award X-063-1014 from the Qatar National Research Fund (a member of The Qatar Foundation). The statements made herein are solely the responsibility of the authors.

data to pass it to the third party. In the following round, that third party works with Carol to execute the next instruction then re-permute and re-encrypt the program and data to pass it back to the second party.

Specifically Alice starts with the encrypted and permuted program and data in addition to the first instruction address. She re-permutes the data and adds 0 to all it's values to change the ciphers. Then she works with Carol to re-permute and re-encrypt the program too. She then executes the first step of the SUBLEQ instruction which alters one data value. Now that the new data has been changed, she can send it together with the new program to Bob for him to use in the next round. She then works with Carol and Bob to perform the second step of the SUBLEQ instruction such that only Bob knows the next instruction to be executed. This step uses SMC specifically Garbled Circuits (GC) [5] to hide the inputs and only output to Bob the next instruction address. At this point a round has finished, the next round can proceed by Bob performing Alice's function and Alice performing Bob's. This back and forth execution repeats until the maximum number of rounds allowed is reached. At that point the last altered data element is decrypted by Carol and returned as the result. If the total number of rounds is not known a priori, the execution can stop when a specific termination address is reached. In this scenario, Carol has to generate the keys dynamically.

III. Analysis

A. Informal Security Analysis

The security model used in this work is a semi-honest model with non-colluding servers.

An instruction execution doesn't leak information. Since data is encrypted using additive homomorphism the first part of a SUBLEQ command can be executed easily using modular multiplication. And since the second part of a SUBLEQ is hidden using SMC, a single instruction execution doesn't leak information.

Program and Data remain hidden throughout rounds. In each round either Alice or Bob will receive $\mathcal{P}$ and $\mathcal{D}$ re-permuted and re-encrypted. Hence they cannot track the instruction traces or memory access traces. Carol on the other hand knows every permutation after the initial ones and holds all the keys, however he doesn't know $\mathcal{P}$ or $\mathcal{D}$ as they are with Alice and Bob (in encrypted form).

B. Performance Evaluation

To understand the computational cost of running a SUBLEQ program, identifying the cost of running a single instruction is enough (total cost is $T\times$ single instruction cost). The bulk of the computation cost in executing an instruction is in the memory permutation. This requires a decryption and an encryption for each argument in the program, a cost linear in the size of the program. In addition, Alice has to compute $|\mathcal{D}|$ encryptions of 0 and homomorphic additions to obfuscate the data, however, these encryptions could be done offline. In total, a single execution round requires about $4|\mathcal{P}| + |\mathcal{D}| + k_0$ encryptions, $4|\mathcal{P}| + k_1$ decryptions and $|\mathcal{D}| + k_2$ homomorphic additions where k_0, k_1 and k_2 are some small constants. The cost of the rest of the operations such as the permutations or the small GC evaluation is negligible.

Communication cost is mainly affected by the size of $\mathcal{D}$ and $\mathcal{P}$. This is because $\mathcal{D}$ and $\mathcal{P}$ need to be sent from Alice to Bob or Bob to Alice every round. Also the program needs to be exchanged between Carol and the current party for re-encryption. Taking into account the blow-up from additive homomorphic encryption cipher size the communication cost is $3 \times 8192|\mathcal{P}| + 2048|\mathcal{D}| + k_3$ bits (k_3 is a small constant that accounts for GC and some other minor details). This cost is per round so it will be repeated T times.

C. Practical Evaluation

An initial system was developed to implement the proposed protocol using C++ and OblivC [6]. Three processes denoting Alice, Bob and Carol run on a single machine with a 3GHz quad core intel Q9650 CPU and a 4GB DDR2 memory.

To evaluate the running time of the system, multiple programs have been tested. One example is fault tree analysis (FTA). FTA is a method used in safety engineering to analyze how a system can fail. It consists of a tree that combines Boolean events to decide whether a failure will occur or not. This method is widely used throughout the industry and in some cases computing the result of fault-tree analysis in a two-party secure way is desirable. FTA represents a realistic application where private data is provided by the machine owner while the private fault analysis program is provided by the machine manufacturer.

For testing, a 5-gate fault tree analysis program was converted into a 15-instruction SUBLEQ code that takes as input 9 data values. The SUBLEQ program ran for 12 rounds ($T = 12$) on an average of 6.5 seconds.

IV. Conclusion

This work presents a novel protocol for Private Function Evaluation using three non-colluding parties. It attempts at being more practical than previous protocols. Future work includes, improving the protocol to reduce the computation and communication cost, and improving the preliminary system implementing it for faster running times.

References

[1] V. Kolesnikov and T. Schneider, "A practical universal circuit construction and secure evaluation of private functions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2008, pp. 83–97.

[2] P. Mohassel and S. Sadeghian, "How to hide circuits in mpc an efficient framework for private function evaluation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 557–574.

[3] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[4] O. Mazonka and A. Kolodin, "A simple multi-processor computer based on subleq," *arXiv preprint arXiv:1106.2593*, 2011.

[5] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

[6] S. Zahur and D. Evans, "Obliv-c: A language for extensible data-oblivious computation." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1153, 2015.