Oblivious Turing Machine

Sofiane Azogagh, Victor Delfour, Marc-Olivier Killijian

Oblivious Turing Machine

Sofiane Azogagh
Computer Science Department
Université du Québec à Montréal
Montréal, Québec, Canada

Victor Delfour*
Computer Science Department
Université du Québec à Montréal
Montréal, Québec, Canada
delfour.victor@courrier.uqam.ca

Marc-Olivier Killijian
Computer Science Department
Université du Québec à Montréal
Montréal, Québec, Canada

***Abstract*—In the ever-evolving landscape of Information Technologies, private decentralized computing on an honest yet curious server has emerged as a prominent paradigm. While numerous schemes exist to safeguard data during computation, the focus has primarily been on protecting the confidentiality of the data itself, often overlooking the potential information leakage arising from the function evaluated by the server. Recognizing this gap, this article aims to address the issue by presenting and implementing an innovative solution for ensuring the privacy of both the data and the program. We introduce a novel approach that combines the power of Fully Homomorphic Encryption with the concept of the Turing Machine model, resulting in the first fully secure practical, non-interactive oblivious Turing Machine. Our Oblivious Turing Machine construction is based on only three hypotheses, the hardness of the Ring Learning With Error problem, the ability to homomorphically evaluate non-linear functions and the capacity to blindly rotate elements of a data structure. Only based on those three assumptions, we propose an implementation of an Oblivious Turing Machine relying on the TFHE cryptosystem and present some implementation results.**

I. Introduction

While cloud computing offers numerous advantages, concerns around data privacy persist. Despite the fact that some businesses may be comfortable with their data being processed in clear by servers in the cloud, most organizations prioritize data privacy for a variety of reasons : ensuring their client confidentiality, protecting intellectual property, and preventing potential competitive disadvantages resulting from data leaks are just a few examples. As a result, there is a growing demand for private cloud computing solutions that ensure data confidentiality even when the server, and potential observers, are aware of the program applied to the data either because it is publicly available or proprietary to the server.

However, it is essential to recognize that the server's knowledge of the program applied to the data can inadvertently reveal sensitive information about the client. For instance, a client requesting the application of a cancer detection model to their data might indicate a strong suspicion of having cancer. This information can be highly personal and should be treated with utmost sensitivity to protect the client's privacy and well-being.

A Turing machine is an abstract model of a theoretical computer that encompasses the capabilities of a general-purpose computing device. It serves as a fundamental tool in computer science and is capable of executing any computation that can be performed by a computer. The strength of a Turing machine lies in its ability to simulate any algorithm, making it a versatile and powerful computational model. The concept of *obliviousness* applied to machines was introduced by Pippenger and Fischer [24] back in 1979. According to their definition, a machine is considered oblivious if an adversary observing the behaviour of the machine, without seeing the content of the cells, can conclude nothing about the inputs of this machine, except from the time required for the machine to perform the evaluation. At that time, ensuring this property of obliviousness with Turing machines meant ensuring that the movements of the head on the tape are the same in execution time for each input and for each step. But nowadays, with fully homomorphic encryption (FHE) we can not only encrypt the contents of the tape, but also the movements of the head rendering them indistinguishable from one step to another. This leads to the introduction of the concept of *fully oblivious* Turing Machines. Such versions aim to enable the private delegation of computations to servers in the semi-honest model while ensuring that no information regarding the algorithm, its inputs or outputs is leaked to the untrusted server, or to potential observers. The significance of achieving fully oblivious computation extends to various domains, including sensitive data analysis, privacy-preserving machine learning, and confidential financial computations. By proving the feasibility of computing without divulging any information about the algorithm or its inputs and outputs, fully oblivious computation ensures a higher level of privacy and security in outsourced computation scenarios. From now on, and to the end of this article, we will refer to fully oblivious Turing Machines simply as Oblivious Turing Machines (OTM).

A. Our contribution

To the best of our knowledge, this work presents a significant advancement in the field of secure computation by introducing the first non-interactive fully oblivious Turing machine. Our scheme enables the computation of any algorithm without divulging any knowledge about the algorithm, its inputs, or its

This work is supported by the DEEL Project CRDPJ 537462-18 funded by the National Science and Engineering Research Council of Canada (NSERC) and the Consortium for Research and Innovation in Aerospace in Québec (CRIAQ), together with its industrial partners Thales Canada inc, Bell Textron Canada Limited, CAE inc and Bombardier inc. (https://deel.quebec) Marc-Olivier Killijian is supported by NSERC through a Discovery Grant as well as the DEEL project. Sofiane Azogagh is funded by the DEEL project, Victor Delfour through the NSERC Discovery Grant.

outputs. With this approach, clients can confidently delegate computation to an honest but curious server, ensuring trust and privacy. The scheme we propose has the following key properties :

- Non-interactivity: our scheme only requires one round of communication between the client and the server.
- Full Obliviousness: both the client's data and function are completely concealed from the server. The server only learns the number of steps of the Turing machine requested by the client, without any additional knowledge.

The elimination of interactivity significantly reduces the communication cost, minimizing it to the essential elements: transmitting the function (Turing machine table of rules), its input (the tape before computation and the initial state), and receiving the final result (the tape and state after computation). Additionally, the reduced interactivity implies a lower computational cost for the client, which is particularly advantageous in cloud computing scenarios where clients aim to minimize their workload. By leveraging the non-interactive fully oblivious Turing machine, our scheme not only provides strong privacy guarantees but also addresses the practical considerations of communication and computational efficiency. These advances contribute to the broader landscape of secure and privacy-preserving cloud computing, enabling clients to confidently delegate computation while minimizing both communication and computational overhead.

In the following sections, we delve into the technical aspects of our scheme, discussing the necessary cryptographic techniques and protocols. We also present some experimental results that prove the feasibility our proposed approach.

B. Related Work

In the field of Secure Computing, the majority of techniques focus on ensuring the confidentiality of the data being processed. Some of them also address the need of proving that computation is correct. However, there are relatively few, if any, approaches that specifically address confidentiality of the program being computed.

Pippenger et al [24] show that any Turing machine of n steps can be made oblivious by performing $\mathcal{O}(nlog(n))$ step on a two-tape machine. However, this does not imply any privacy of the inputs or of the machine itself. Komargodski et al [18] extend this concept with the introduction of the first Differentially Oblivious Turing Machine, adding differential privacy to the movements of the heads of the machine. Following the work from Pippenger et al, Goldreich and Ostrovsky introduced Oblivious RAM (ORAM) [13], [23]. ORAM provides a functionality between a client and a server, allowing the client to read and modify data in the server's database without the server knowing the specific memory access path chosen by the client. Over the past 15 years, ORAMs have been extensively researched and can be utilized to create an Oblivious Turing Machine with high interactivity [20].

Of course, Homomorphic Encryption (HE), whether fully or somewhat homomorphic, is a notable technique that allows a client to delegate computation to a server while operating on encrypted data. The server performs computation on this encrypted data and produces an encrypted result. HE ensures that the server never handles clear (unencrypted) data, but it is aware of the algorithm being executed. However, it is possible to have a more subtle approach using HE and to make the algorithm oblivious to the server as we propose to do here.

Functional Encryption (FE) is a powerful technique that enables computations to be performed on encrypted data by deriving decryption keys specific to a particular function within a family of functions. While much of the research in FE has focused on preserving the privacy of the encrypted data, recent works have started exploring the privacy of the function itself. One notable contribution in this field is the work by Bishop et al. [8], which introduces the first collusion-resistant and function-private FE scheme specifically designed for inner product functionality based on standard assumptions. Another significant advancement is presented by Kim et al. [17], who propose an enhanced protocol for inner product encryption based on the decisional linear (DLIN) assumption. This protocol offers a small master secret key size and efficient decryption and key generation algorithms. Ananth et al [2] proposed a scheme to preserve the privacy of a Turing Machine being computed through the use of Functional Encryption. However, this scheme does not protect the input tape.

Private Function Evaluation (PFE) is a cryptographic primitive that uses HE to enable the delegated evaluation of a private function detained by one party on private inputs coming from some other(s) party(es), on a honest but curious server. In the case of a malicious server, there is a high detection probability as the output might fall outside the function's range. PFE schemes primarily aim to reduce communication cost, round complexity, and improve efficiency of the private computation of a function. Several protocols based on boolean circuits have been proposed, for example [16], [21], [22], [7]. For example, the scheme proposed by Biçer et al. [7] is a two-party private function evaluation scheme for boolean circuits, designed to provide security in the semi-honest model. This scheme achieves cost reductions by reusing tokens in the two-party computation stage, thereby eliminating redundant computations and message exchanges during subsequent evaluations of the same function. Unlike with the general delegated computing model we aim for with our Oblivious Turing Machine, PFE schemes are targeting specific cases where one party possesses the program and another has, or several others have, the inputs.

Indistinguishability Obfuscation (iO) takes the program obliviousness one step further in the sense that it aims to obfuscate programs in a manner where functionally equivalent programs become computationally indistinguishable [5]. The primary objective of iO is to safeguard the internal logic of a program, rendering it practically impossible for an observer to discern its functionality. A line of research has focused on adapting the principles of iO to the Turing Machine model [19], [12], [1], [4]. However, these works employ circuits in a way that ties the machine's encryption to its input and, at times, the number of steps. In contrast, our construction

independently encrypts the machine, regardless of its input, and is applicable for an unbounded number of steps, ensuring reusability at will. Our scheme offers a novel approach to iO, as the indistinguishability of our programs relies solely on the indistinguishability of homomorphic ciphertexts.

Non-Interactive Oblivious Turing Machines were introduced by Rass [25]. In their work, they provide a scheme that simulates a Turing Machine under somewhat homomorphic encryption. However, this proposal operates under homomorphic public-key encryption with an equality check (HPKEET). The equality test outputs the result of the comparison as a plaintext, which defeats the purpose of homomorphic encryption : an honest but curious server can easily determine when two ciphertexts are encryptions of the same plaintext, thus negating the indistinguishability between ciphertexts. A later proposal by Rass et al [26] attempts to fix this flaw by using a trusted hardware environment. An implementation of this scheme was proposed by Dutra et al [11].

An innovative approach to Non-Interactive Oblivious Turing Machines has been proposed by Goldwasser et al [14], [15]. They propose schemes that aim to run Turing Machines in a decentralized manner using Attribute-Based Encryption and Functional Encryption. This work is the first of its kind to present an OTM construction without an obvious flaw in the security and without trusted hardware. Their schemes rely on a large number of cryptographic components such as SNARKs, Witness Encryption, Functional Encryption, Signatures and FHE. As a consequence, their schemes are based on several assumptions such as Learning With Errors (LWE), Decision Graded Encoding No-Exact-Cover (DGE No-Exact-Cover) and knowledge of exponent assumption (KEA1). This abundance of cryptographic components and assumptions is not desirable as it leads to complex schemes with more risks of attacks. In the contrary, our scheme only rely on the possibility to homomorphically evaluate non-linear functions, blind rotations and the RLWE problem hardness. Moreover, their scheme uses the Pippenger-Fisher transformation from [24] which increases the number of Turing Machine steps from n to $O(nlog(n))$. To the best of our knowledge, there is no implementation of these proposals.

II. Preliminaries

In this section, we introduce all the notions that will be necessary for the understanding of our proposed scheme : we define what we call a Turing Machine (TM) and an Oblivious Turing Machine, then we present the TFHE homomorphic encryption scheme along some notations and its specific features we rely on in our construction, and finally by introducing a new primitive to blindly access matrices.

A. Turing Machines

As stated above, a TM is an abstract, yet complete, computing model. A TM is composed of some logic table controlling a state, a tape and a head. The tape is a data structure of infinite length, consisting of cells that hold symbols from a specific alphabet. This is where the input data is written and where the output is generated at the end of the computation. The head is responsible for reading and writing the cells of the tape one at a time. It has a specific position that can change by at most one cell between consecutive time steps, i.e. it can move left, right or stay at the same position of the tape. At each time step, the TM executes the following sequence of actions: it first reads the cell at the current head position, it potentially overwrites the cell, and possibly moves the head and changes its state according to its current state, the symbol read and its table of instructions. The state of a TM is represented by an integer, indicating which line of the machine's instructions the machine should refer to after reading the cell under the head. There is a special state known as the final state, which indicates whether or not the evaluation is complete. The table of instructions is a finite table that encodes the function being computed. Each line in this table contains instructions for one time step, based on both the current state and the symbol under the head. The instructions specify what the head should write, where it should move, and what the next state will be.

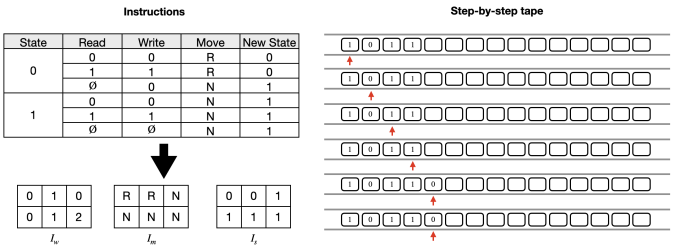

Figure 1. An illustration of a classic Turing Machine (TM) designed to perform binary multiplication by 2 (with the alphabet being $\mathbb{B}$ and the blank symbol). The red arrow represents the head of the TM. The machine's instructions are documented in a table that can be expressed as three matrices: I_w for writing instructions, I_m for moving instructions, and I_s for state-changing instructions. The behavior of the TM will depends on the cell content under its head and the current state.

In this description, there are several pieces of information that the client could want to conceal from the server. First, the client wants to ensure that the server gains no information about the data being processed, except for an upper bound on its size, i.e. the length of the tape. To achieve this, the data needs to be homomorphically encrypted. Second, there is information related to the implementation of the function. The function is represented by the table of instructions, which also needs to be homomorphically encrypted using the same keys as the data. This allows for direct computation on the encrypted data. The server only obtains an upper bound on the size of the function, which is directly related to the size of the table of instructions. Finally, the most challenging part is hiding the data associated with the function's execution from the server. To ensure that the server gains no information, each step of the TM needs to be indistinguishable from any other step. Achieving this indistinguishability relies on the ability to blindly access the instruction table, which is explained in section II-C, and which is based on some specific TFHE features that we present below.

B. TFHE Scheme

The TFHE scheme, introduced in [9], [10], is a Fully Homomorphic Encryption (FHE) scheme. Its security is based on the Learning With Errors (LWE) problem and its variant, Ring Learning With Errors (RLWE). In TFHE, there are three types of ciphertexts. We denote as p the message space size.

1) **LWE ciphertexts**, which represent an encrypted element from the ring $\mathbb{Z}_p$. LWE ciphertexts naturally support the addition operation between ciphertexts and also the absorption operation, a multiplication between a ciphertext and a cleartext.
2) **RLWE ciphertexts**, on the other hand, are the ring version of LWE ciphertexts in the sense that they are an encryption of a polynomial in $\mathcal{R}_p$[1]. The advantage of these ciphertexts over LWE is that they allow several elements to be processed at the same time. In TFHE, RLWE ciphertexts also support addition and absorption.
3) **RGSW ciphertexts** are two-dimensional matrices of RLWE ciphertexts. These ciphertexts enable an external product operation. This operation takes as inputs a RGSW ciphertext and a RLWE ciphertext, and outputs a RLWE ciphertext holding the product of the messages encrypted in the inputs.

.

Beside these three types of ciphertexts, we introduce **LUT ciphertexts** which are RLWE ciphertexts that include some redundancy. More formally, while a RLWE ciphertext is an encryption of $\mu(X) = \sum_{i=0}^{N-1} m_i X^i$ a LUT ciphertext is an encryption of $M(X) = \sum_{i=0}^{p-1} \sum_{j=i}^{(i+1)\Delta-1} m_i X^j$ where $\Delta = \frac{N}{p}$. In the following, ciphertexts will be denoted within brackets, indicating the type of ciphertext used the first time it is introduced. For example, $[\![m]\!]_{\mathrm{LWE}}$ represents the message m encrypted as an LWE ciphertext and will be referred to as $[\![m]\!]$ beyond the first encounter.

The TFHE cryptosystem includes several useful functions that are relevant to our construction. We provide here a brief description of these functions:

- **Sample Extraction (SE)**: $(\star, [\![\star]\!]_{\mathrm{RLWE}}) \longrightarrow [\![\star]\!]_{\mathrm{LWE}}$ This function allows the extraction of a single coefficient from an RLWE ciphertext. The extracted coefficient is represented as a LWE ciphertext. The position of the coefficient is not encrypted and is known to the party performing the operation.
- **Blind Rotation (BR)** : $([\![\star]\!]_{\mathrm{RLWE}}, [\![\star]\!]_{\mathrm{LWE}}) \longrightarrow [\![\star]\!]_{\mathrm{RLWE}}$ This function takes an LWE ciphertext and an RLWE ciphertext as input. It produces an RLWE ciphertext as output, where the polynomial under the RLWE input is privately rotated by the value encrypted in the LWE ciphertext. The rotation is performed without revealing any information about the original polynomial.
- **Public Functional Key Switch (PFKS)** : $([\![\star]\!]_{\mathrm{LWE}}, \ldots, [\![\star]\!]_{\mathrm{LWE}}) \longrightarrow [\![\star]\!]_{\mathrm{RLWE}}$ This feature introduced in [10] (Alg. 2) enables the compact representation of multiple LWE ciphertexts into a single RLWE ciphertext. It takes several LWE ciphertexts as input and packs them into a RLWE ciphertext.

[1]For N a power of 2, we denote by $\mathcal{R}$ the quotient ring $\mathbb{Z}[X]/(X^N+1)$ and by $\mathcal{R}_q$ the same ring modulo q.

The first two operations, SE and BR, are part of the *bootstrapping* process of TFHE. In a nutshell, bootstrapping allows the reduction of the noise in a ciphertext, avoiding the message from being overlapped by it, which would lead to a decryption error. This is the core and the most expensive operation of FHE schemes. The specificity of the TFHE scheme is that its bootstrapping allows to evaluate any discretized function. This is known in the literature as *functional bootstrapping* (FB) or *programmable bootstrapping* (PBS).

The function evaluated with PBS is discretized into a Look Up Table (LUT). On a high level, PBS takes as input a LWE ciphertext $[\![x]\!]_{\mathrm{LWE}}$ and a LUT of a function f encrypted in a RLWE ciphertext, then it returns a LWE ciphertext $[\![f(x)]\!]_{\mathrm{LWE}}$. Evaluating a function in that way comes with a cost in precision. This means that when we want to encrypt a LUT of f in a RLWE ciphertext, we have to add redundancy to each element of the LUT for noise management purposes. We will then denote this specific encoding as $[\![f]\!]_{\mathrm{LUT}}$.

C. Accessing an encrypted matrix blindly

An important element of our construction, which may be of independent interest, is what we call *Blind Matrix Access* (BMA). It is the extension of *Blind Array Access* (BAA) introduced in [3] which enables a server to obliviously access a one-dimensional encrypted array represented as a RLWE ciphertext. The access consists in performing a PBS on the array with the input index to be accessed. Regarding multi-dimensional arrays, instead of having a single RLWE ciphertext, we have a vector of RLWE ciphertexts. This vector will allow us to output the column as a vector of LWE ciphertexts by computing the programmable bootstrap for each of those RLWE ciphertexts at the encrypted index of the column. Once this vector is obtained, it is transformed into a RLWE ciphertext, using the Public Functional Key Switching from [10]. Finally, we obtain a single RLWE ciphertext and the problem is reduced to one dimension, so we can use a simple *Blind Array Access* to extract the correct coordinate. The algorithm 1 captures the behavior of BMA.

This algorithm allows obtaining an element from an matrix M based on its indices i and j. The matrix M is encoded and encrypted as a vector of LUT, which represents the rows of M. If the number of rows is less than p, the client must complete the matrix with p rows to conceal this information, which can be sensitive depending on the use case. By doing so, the computational cost of Blind Matrix Access will depend only on p and will be constant for all matrices. However, if the number of rows is not sensitive, then not completing the matrix up to p rows can significantly optimized BMA accesses as illustrated in 2.

Algorithm 1 BlindMatrixAccess (BMA)

Input : $M = (\llbracket r_0 \rrbracket_{\text{LUT}}, \ldots, \llbracket r_{p-1} \rrbracket_{\text{LUT}})$ the matrix to be accessed, $\llbracket i \rrbracket_{\text{LWE}}$ the index's rows, $\llbracket j \rrbracket_{\text{LWE}}$ the index's column, bk the bootstrapping key, p the message space

Ensure: $\llbracket M[i][j] \rrbracket_{\text{LWE}}$

function BLINDMATRIXACCESS($M, \llbracket i \rrbracket_{\text{LWE}}, \llbracket j \rrbracket_{\text{LWE}}$)

for $s = 0 \ldots p-1$ **do**

$\llbracket c_s \rrbracket_{\text{LWE}} \leftarrow$ **BAA**$(\llbracket r_s \rrbracket_{\text{LUT}}, \llbracket i \rrbracket_{\text{LWE}}, bk)$ ▷ Alg 3 of [3]

end for

$\llbracket c \rrbracket_{LUT} \leftarrow$ **PFKS**$(\llbracket c_0 \rrbracket_{\text{LWE}} \ldots \llbracket c_{p-1} \rrbracket_{\text{LWE}})$ ▷ Alg 2 of [10]

$\llbracket m_{i,j} \rrbracket_{\text{LWE}} \leftarrow$ **BAA**$(\llbracket c \rrbracket_{\text{LUT}}, \llbracket j \rrbracket_{\text{LWE}}, bk)$

return $\llbracket m_{i,j} \rrbracket_{\text{LWE}}$

end function

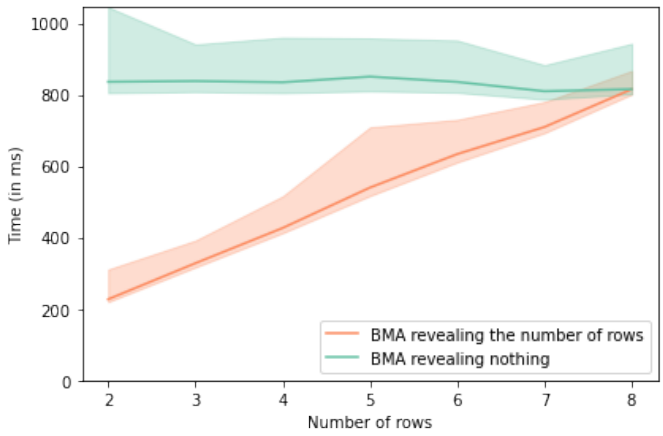

Figure 2. The comparison of Blind Matrix Access running times with and without disclosing the number of rows, using a message space of $p = 8$.

III. Our construction

A. General overview

The execution of the Turing machine involves a sequence of steps performed one after another. Each step involves reading the content of the cell under the head, updating the content of the cell, modifying the position of the head, and updating the state. In this section, we describe how to perform these operations obliviously and without interaction with the client. We first present the representation of the TM components, and then explain how they work together.

The tape of a TM is conceptually an infinite sequence of integers, although obviously in practice it is finite. In our construction, we represent the tape as a LUT encrypted as a RLWE ciphertexts for several reasons. First, a RLWE ciphertext encrypts a polynomial from the ring $\mathcal{R}_q$, where the coefficients can be regarded as the elements on the TM tape. Second, thanks to TFHE we can perform blind rotations on RLWE ciphertexts, which allows us to make the position of the head intrinsic to the tape itself. By blindly rotating the tape at each step, we can consider that the head remains stationary while the tape moves underneath it. For instance, in our scheme, the head is located at the first element of the LUT. Then to move it to the left, we perform a blind rotation of the tape by -1 (or more accurately, by $2p-1$, to account for the negacyclicity inherent in the ring $\mathcal{R}_q$). Conversely, to move it to the right, we rotate it by 1, and for no movement at all, we perform a rotation of 0. The direction is determined by the program instruction table. This approach enables to always focus on the first element of the LUT for both reading and writing, while maintaining obliviousness throughout the process as any element of the tape could be positioned under the head due to the oblivious execution.

The state of the TM is encrypted as a LWE ciphertext. It is updated at every step to ensure that any change or lack of change in the state remains oblivious to the server. The table of instructions is decomposed into three matrices as illustrated in Figure 1. The I_w matrix determines what is to be written on the tape, the I_m matrix describes the head (the tape) moves, and the I_s matrix governs state updates. The server blindly accesses these matrices using *Blind Matrix Access*. The three matrices are statically determined by the client before execution, thus acting like a ROM.

B. Description of a step

First, the server reads the content of the tape by calling *Blind Array Access* at index 0 (the index is a trivial encryption of 0). This operation allows the server to get the element under the head $\llbracket c \rrbracket_{\text{LWE}}$ which will be the column index of the wanted element in the encrypted matrices $\llbracket I_w \rrbracket$, $\llbracket I_m \rrbracket$, $\llbracket I_s \rrbracket$ while the current state $\llbracket s \rrbracket_{\text{LWE}}$ will be its row index. Then the server overwrites the content of the cell under the head with the value determined by $\llbracket I_w[s] \rrbracket_{LWE}$, which actually encodes the value to be added to the current cell content to obtain the desired result. Then, the TM moves the tape to simulate a change in the head's position in the direction given by $\llbracket I_m[s] \rrbracket$ using a *Blind Rotation* operation. Finally it updates its state to $I_s[s]$, this state will be the one used in the next step. Once all the requested number of steps have been completed, the server sends the tape and the state to the client. This is described in Algorithm 2, the main algorithm of our Turing Machine.

Algorithm 3 describes reading the tape content, it extracts the first element of the tape using *Blind Array Access*. Writing the new content of the cell is described in Algorithm 4 : it gets an LWE ciphertext encrypting the element to be added to the current content in order to determine the value to be written at the cell under the head. However, due to the tape being a LUT encrypted as an RLWE ciphertext, adding it with the LWE ciphertext requires a conversion of the LWE to a RLWE ciphertext incorporating the appropriate redundancy to form a valid LUT ciphertext. This is done using the Private Functional Key Switching algorithm from [10].

Algorithm 5 updates the tape by blindly rotating it either by -1, 0, or 1 it through the *Blind Rotation* algorithm, effectively changing the position of the head. We track the head's movements by consistently updating an encrypted counter. This counter, by the end of Algorithm 2, will be utilized to restore the head to its first position.

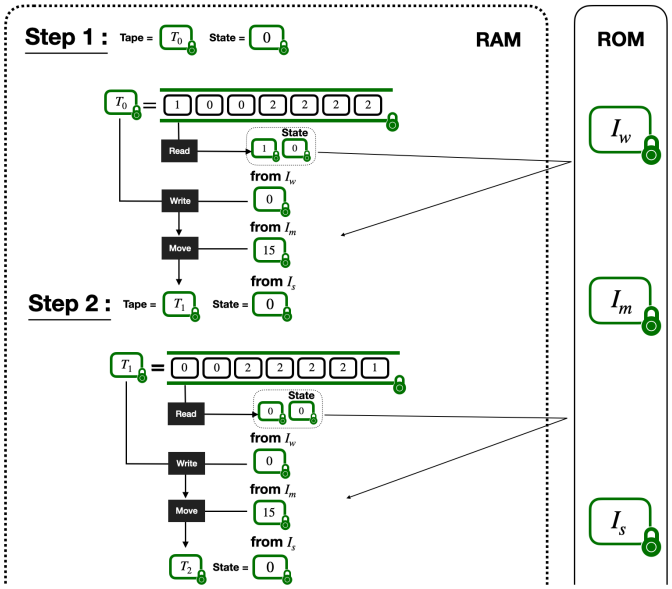

Figure 3. The operations of the first two steps of the Turing machine. The initial phase of each step involves blindly reading the cell under the head, employing the Blind Array Access algorithm outlined in [3]. Subsequently, the resulting ciphertext, combined with the current state, is utilized to access the three matrices: I_w, I_m, and I_s. These matrices respectively dictate what to write in the cell using Algorithm 4, where to position the head for the next step using Algorithm 3, and the new state to be utilized in the subsequent step using Algorithm 6.

Algorithm 2 Oblivious Turing Machine (OTM)

Input : $[\![T]\!]_{\text{LUT}}$ the tape, $[\![s]\!]_{\text{LWE}}$ the current state, $[\![I]\!]$ the instruction table as a tensor such as $I = (I_w, I_m, I_s)$ and n the number of steps
Ensure: The tape $[\![T]\!]$ containing the result of the evaluation
function OTM($[\![T]\!]_{\text{LUT}}$,$[\![s]\!]_{\text{LWE}}$,$[\![I]\!]$,n)
  $[\![\ell]\!]_{\text{LWE}} \leftarrow [\![0]\!]_{\text{LWE}}$ ▷ Counter to track the head
  for $i = 1$ to n **do**
    $[\![c]\!]_{\text{LWE}} \leftarrow$ **Read**$([\![T]\!]_{\text{LUT}})$
    Write$([\![T]\!]_{\text{LUT}}, [\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}}, [\![I]\!])$
    Move$([\![T]\!]_{\text{LUT}}, [\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}}, [\![I]\!], [\![\ell]\!]_{\text{LWE}})$
    $[\![s]\!]_{\text{LWE}} \leftarrow$ **GetNewState**$([\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}}, [\![I]\!])$
  end for
  $[\![T]\!]_{\text{LUT}} \leftarrow$ **BR**$([\![T]\!]_{\text{LUT}}, [\![\ell]\!]_{\text{LWE}})$ ▷ Replacing the head
  return $[\![T]\!]$
end function

Algorithm 3 Read

Input : $[\![T]\!]_{\text{LUT}}$ the tape
Ensure: $[\![c]\!]_{\text{LWE}}$ the content of the cell under the head
function READ($[\![T]\!]_{\text{LUT}}$)
  $[\![c]\!]_{\text{LWE}} \leftarrow$ **Sample Extract**$([\![T]\!]_{\text{LUT}}, 0)$
  return $[\![c]\!]_{\text{LWE}}$
end function

Algorithm 4 Write in place (Write)

Input : $[\![T]\!]_{\text{LUT}}$ the tape, $[\![c]\!]_{\text{LWE}}$ the cell content, $[\![s]\!]_{\text{LWE}}$ the current state, $[\![I]\!]$ the instruction table
Ensure: The tape $[\![T]\!]_{\text{LUT}}$ is updated
function WRITE($[\![T]\!]_{\text{LUT}}, [\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}}, [\![I]\!]$)
  $[\![w]\!]_{\text{LWE}} \leftarrow$ **BMA**$([\![I]\!], [\![s]\!]_{\text{LWE}}, [\![c]\!]_{\text{LWE}})$
  $[\![W]\!]_{\text{LUT}} \leftarrow$ **PFKS**$([\![w]\!]_{\text{LWE}}, \ldots, [\![w]\!]_{\text{LWE}})$ ▷ $\frac{N}{p}$ entries
  $[\![T]\!]_{\text{LUT}} \leftarrow [\![T]\!]_{\text{LUT}} + [\![W]\!]_{\text{LUT}}$
end function

Algorithm 5 Move in place (Move)

Input : $[\![T]\!]_{\text{LUT}}$ the tape, $[\![c]\!]_{\text{LWE}}$ the cell content, $[\![s]\!]_{\text{LWE}}$ the current state, $[\![I]\!]$ the instruction table, $[\![\ell]\!]_{\text{LWE}}$ the counter to track the head movements
Ensure: The tape $[\![T]\!]_{\text{LUT}}$ is moved and the counter $[\![\ell]\!]_{\text{LWE}}$ is updated.
function MOVE($[\![T]\!]_{\text{LUT}}, [\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}}, [\![I]\!], [\![\ell]\!]_{\text{LWE}}$)
  $[\![m]\!]_{\text{LWE}} \leftarrow$ **BMA**$([\![I]\!], [\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}})$
  $[\![T]\!]_{\text{LUT}} \leftarrow$ **BR**$([\![T]\!]_{\text{LUT}}, [\![m]\!])$ ▷ Alg 4 of [10]
  $[\![\ell]\!]_{\text{LWE}} \leftarrow [\![\ell]\!]_{\text{LWE}} + [\![m]\!]_{\text{LWE}}$ ▷ Track the head
end function

Algorithm 6 GetNewState

Input : $[\![T]\!]_{\text{LUT}}$ the tape, $[\![c]\!]_{\text{LWE}}$ the cell content, $[\![s]\!]_{\text{LWE}}$ the current state, $[\![I]\!]$ the instruction table, $[\![\ell]\!]_{\text{LWE}}$ the counter to track the head movements
Ensure: $[\![s]\!]_{\text{LWE}}$ the new state of the Turing Machine
function GETNEWSTATE($[\![T]\!]_{\text{LUT}}, [\![c]\!]_{\text{LUT}}, [\![s]\!]_{\text{LUT}}, [\![I]\!]$)
  $[\![s]\!] \leftarrow$ **BMA**$([\![I]\!], [\![c]\!]_{\text{LWE}}, [\![s]\!]_{\text{LWE}})$
  return $[\![s]\!]_{\text{LWE}}$
end function

Algorithm 6 gets the new state of the Turing Machine for the next step based on the current state and the content of the cell under the head.

After executing the specified number of steps, the server sends back ciphertexts of the tape and the current state to the client, regardless of whether or not the Turing Machine has finished its computation. In order to determine if the TM has reached its final state, the client can decrypt the output. If the computation is not complete, the client can estimate the number of steps required to terminate and request the server to restart a computation for that number of steps. It is important to note that, from the point of view of the server, this will look like a fresh new request by the client and there is no way for it to learn that it is a continuation of the previous run. Unlike with traditional cloud computing, where a client may pay for a certain amount of computation time and receive no output if the algorithm does not terminate, this construction guarantees that the client receives some intermediary result within the specified number of steps. These properties provide efficiency and reliability in the computation process, allowing the client

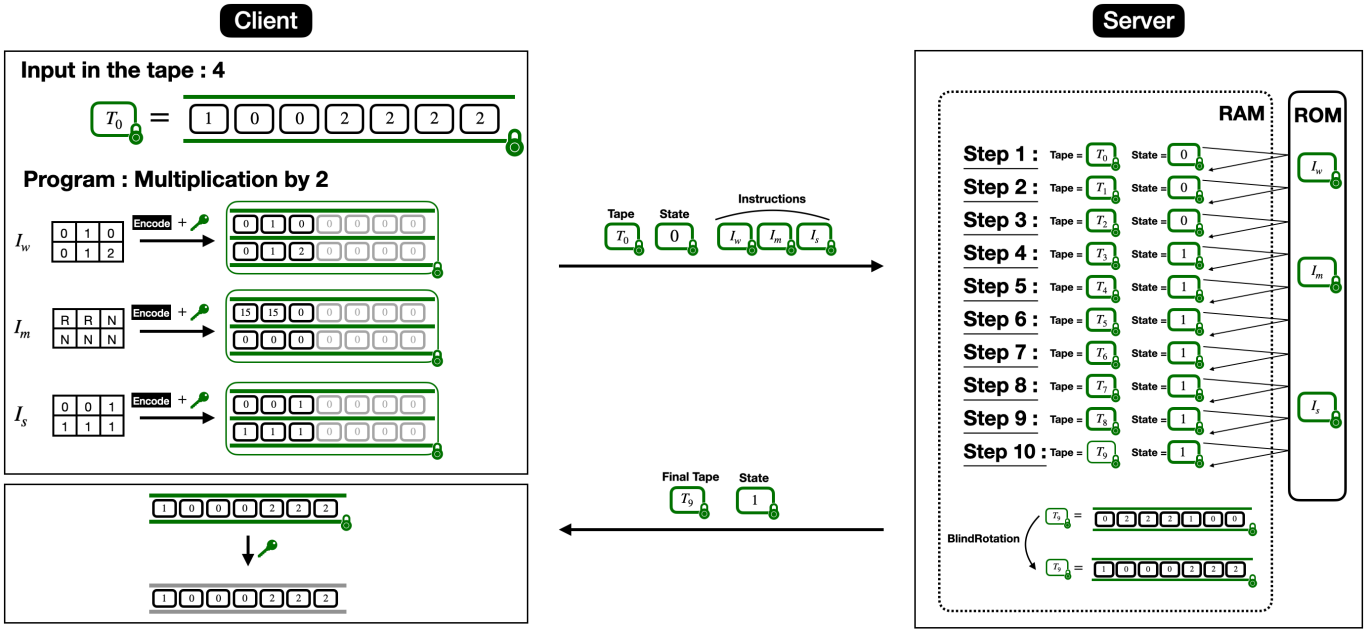

Figure 4. An illustration of the complete pipeline for our one-round Oblivious Turing Machine protocol. Initially, the client encrypts the tape containing the input using LUT ciphertext, encrypts the current state with LWE ciphertext, and represents the instruction matrices as a vector of LUT ciphertext. These ciphertexts are then transmitted to the server, which executes the number of steps specified by the client. In each step, the three static matrices, as described in 3, are used to update the state, the tape, and an encrypted counter that tracks the movement of the heads. After completing the steps process, the server restores the head to its original position by performing a blind rotation of the tape. Subsequently, the server sends back the final tape and the current state to the client who decrypt them to obtain the result.

to make informed decisions based on the intermediary results and continue the computation if needed.

IV. Results and Discussion

We implemented our scheme of a non-interactive fully oblivious Turing Machine on Rust, using the tfhe-rs library [27]. Our implementation is publicly available on Github [2]. The most important result is certainly the fact that this is the first implementation of such an oblivious TM, while previous work were either non-implemented and probably non implementable or they were interactive. The performance results presented in Fig. 5 were obtained on an Apple MacBookPro with an M1 chip and 16 GB of RAM. This shows the running time of one step of the TM depending of the parameters and whether or not we chose to leak the size of the instruction table. We can remark that when we choose to not leak this information, the running time is simply the maximum running time of these parameters when compared to the non leaky curve. Yet, we believe that disclosing the exact size of the instruction table is, in most application cases, an acceptable leakage. It is also worth noting that for applications with a number of symbols smaller than the number of states, it is beneficial to transpose the TM matrices.

The results shown in Figure 5 results look similar to those of Figure 2 as a step running time comes essentially from 3 BMA and the cost of a BMA is close to the LWE ciphertexts packing cost implied by PFKS and any optimisation of this scheme

[2]https://github.com/sofianeazogagh/ObliviousTM

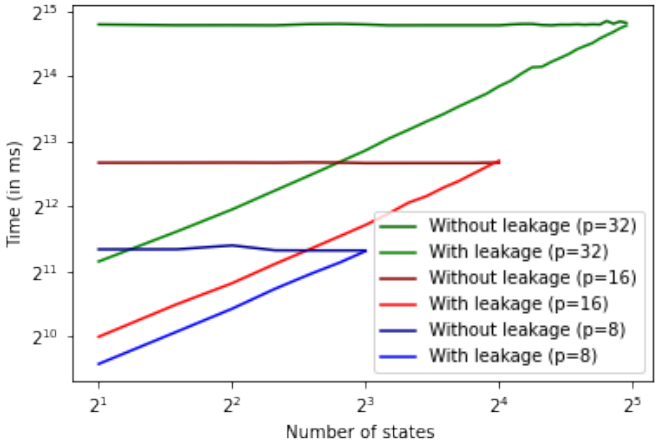

Figure 5. Comparison of OTM step running time with and without revealing the number of state for various message space sizes ($p \in 8, 16, 32$)

would lead to a significant enhancement of our performances. The selection of cryptographic parameters directly influences the tape's length, the potential symbols, and the possible states. Consequently, if the chosen cryptographic parameter for TFHE implies a precision of n bits, these three aforementioned quantities will be constrained by 2^n. Therefore, a user must evaluate the computational and space complexity of their program before determining the cryptographic parameters. For instance, for a given alphabet $\mathcal{A}$ that contains l symbol, if the client aims to evaluate a function f with a domain in $\mathcal{A}^n$ and a co-domain in $\mathcal{A}^m$ that involves k states, he should

parameterize for a precision of max(l^n,l^m,k).

While our construction provides a solid foundation for oblivious computation, there are still opportunities for further research and improvement. One avenue for future work is to explore optimizations to reduce the computational cost of the homomorphic operations involved in the construction. This would enhance the efficiency of the overall system and enable it to handle larger and more complex computations. Indeed, several optimization in the use of the machine can be considered such as using a larger alphabet than binary as it would reduce the number of steps with larger matrices and cryptographic parameters as trade off.

The relationship between OTM and iO is also an interesting field to explore as our OTM scheme guarantees iO but it might also be possible to construct an OTM from iO. Another interesting question would be to explore two-tapes Turing Machines and its consequences on the performance of the implementation in the oblivious settings. Would the use of several tape decrease the size of the instruction table enough to have smaller cryptographic parameters and henceforth enable better performance ?

V. Conclusion

In the domain of privacy-enhancing technologies, several protocols aim to protect both the program and input from leaking in delegated, or decentralized, computation. However, these schemes suffer from several drawbacks : some of them are specific to certain types of program, some other encrypt the input and program together, thereby preventing the program from evaluating on another input, and most of them imply interactivity between the client and the server. The rare schemes that avoid these fallacies are quite complex and rely on numerous cryptographic primitives and assumptions, making them impractical, and they remain unimplemented. For some of them, their security does not even hold. Our construction of an Oblivious Turing Machine allows for secure and private general computation without client interaction or any of these drawbacks. By representing the tape, state, and instructions as RLWE ciphertexts, we achieve obliviousness in machine execution, relying solely on the RLWE assumption and on the TFHE scheme, resulting in a simple and practical construction, the first of its kind to be implemented.

References

[1] Shweta Agrawal and Monosij Maitra. FE and iO for turing machines from minimal assumptions. In Beimel and Dziembowski [6], pages 473–512.

[2] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Heidelberg, January 2016.

[3] Sofiane Azogagh, Victor Delfour, Sébastien Gambs, and Marc-Olivier Killijian. Probonite: Private one-branch-only non-interactive decision tree evaluation. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 23–33, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Saikrishna Badrinarayanan, Rex Fernando, Venkata Koppula, Amit Sahai, and Brent Waters. Output compression, MPC, and iO for turing machines. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 342–370. Springer, Heidelberg, December 2019.

[5] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[6] Amos Beimel and Stefan Dziembowski, editors. *TCC 2018, Part II*, volume 11240 of *LNCS*. Springer, Heidelberg, November 2018.

[7] Osman Bicer, Muhammed Ali Bingol, and Mehmet Sabir Kiraz. Highly efficient and reusable private function evaluation with linear complexity. Cryptology ePrint Archive, Report 2018/515, 2018. https://eprint.iacr.org/2018/515.

[8] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 470–491. Springer, Heidelberg, November / December 2015.

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016. https://eprint.iacr.org/2016/870.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

[11] Rafael Dutra, Benjamin Mehne, and Jay Patel. Blindtm—a turing machine system for secure function evaluation, 2015.

[12] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Beimel and Dziembowski [6], pages 425–454.

[13] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In Alfred Aho, editor, *19th ACM STOC*, pages 182–194. ACM Press, May 1987.

[14] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. Cryptology ePrint Archive, Report 2013/229, 2013. https://eprint.iacr.org/2013/229.

[15] Shafi Goldwasser, Yael Tauman Kalai, Raluca A Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Overcoming the worst-case curse for cryptographic constructions. *IACR Cryptol. ePrint Arch.*, 2013:229, 2013.

[16] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. Cryptology ePrint Archive, Report 2010/528, 2010. https://eprint.iacr.org/2010/528.

[17] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 544–562. Springer, Heidelberg, September 2018.

[18] Ilan Komargodski and Elaine Shi. Differentially oblivious turing machines. *Cryptology ePrint Archive*, 2022.

[19] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.

[20] Tarik Moataz, Erik-Oliver Blass, and Travis Mayberry. Chf-oram: a constant communication oram without homomorphic encryption. *Cryptology ePrint Archive*, 2015.

[21] Payman Mohassel and Saeed Sadeghian. How to hide circuits in MPC: An efficient framework for private function evaluation. Cryptology ePrint Archive, Report 2013/137, 2013. https://eprint.iacr.org/2013/137.

[22] Payman Mohassel, Saeed Sadeghian, and Nigel P. Smart. Actively secure private function evaluation. Cryptology ePrint Archive, Report 2014/102, 2014. https://eprint.iacr.org/2014/102.

[23] Rafail Ostrovsky. Efficient computation on oblivious RAMs. In *22nd ACM STOC*, pages 514–523. ACM Press, May 1990.

[24] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

[25] Stefan Rass. Blind turing-machines: Arbitrary private computations from group homomorphic encryption. *arXiv preprint arXiv:1312.3146*, 2013.

[26] Stefan Rass, Peter Schartner, and Monika Brodbeck. Private function evaluation by local two-party computation. *EURASIP Journal on Information Security*, 2015:1–11, 2015.

[27] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.