Reusable Garbled Turing Machines without FHE ⋆

Yongge Wang[1] and Qutaibah M. Malluhi[2]

[1] Department of Software and Information Systems, UNC Charlotte, 9201 University City Blvd., NC 28223, USA yonwang@uncc.edu
[2] Department of Computer Science and Engineering, Doha, Qatar qmalluhi@qu.edu.qa

Abstract. Since Yao introduced the garbled circuit concept in 1980s, it has been an open problem to design efficient reusable garbled Turing machines/circuits. Recently, Goldwasser et al and Garg et al answered this question affirmatively by designing reusable garbled circuits and reusable garbled Turing machines. Both of these reusable garbling schemes use fully homomorphic encryption (FHE) schemes as required building components. Here, we use multilinear maps to design a reusable Turing machine garbling scheme that will not need any FHE schemes. Though it is not clear whether our multilinear map based garbling approach could be more efficient than FHE based garbling approach, the goal of this paper is to develop alternative techniques for resuable garbling schemes to stimulate further research in this direction.

1 Introduction

Yao [24] introduced the garbled circuit concept which allows computing a function f on an input x without leaking any information about the input x or the circuit used for the computation of $f(x)$. Since then, garbled circuit based protocols have been used in numerous places and it has become one of the fundamental components of secure multi-party computation protocols. Yao's garbled circuits could be used to evaluate the circuit on one input value only.

Since Yao's work in 1980s, it has been an open problem to design efficient reusable garbled Turing machines. Traditionally, a Turing machine M is first converted to a circuit C_M which is then converted to a garbled circuit $\overline{C}_M$ using Yao's technique. However, using a garbled circuit to evaluate an algorithm on encrypted data takes the worst-case runtime of the algorithm on all inputs of the same length since Turing machines are simulated by circuits via unrolling loops to their worst-case runtime, and via considering all branches of a computation. It is preferred that the runtime of the garbled algorithm on garbled input $\overline{x}$ (of x) should be approximately the same as that of the corresponding un-garbled algorithm on input x. To be more specific, the open problem is to design garbled Turing machines that are efficient from following two aspects: (1) the garbled Turing machine $\overline{M}$ has smaller size than $\overline{C}_M$; (2) For each input x, the evaluation of $\overline{M}$ on $\overline{x}$ takes approximately the same time that M takes on x. In this paper, we answer this open problem affirmatively by showing that for each Turing machine M,

⋆ The work reported in this paper is supported by Qatar Foundation Grants NPRP8-2158-1-423 and NPRP X-063-1-014.

we can construct a reusable garbled Turing machine $\overline{M}$ approximately the same size of M without using fully homomorphic encryption schemes.

Recently, Goldwasser et al [17] and Garg et al [12] constructed reusable garbled circuits by using techniques of computing on encrypted data such as fully homomorphic encryption (FHE) schemes and attribute-based encryption (ABE) schemes for arbitrary circuits. Goldwasser et al [16] also constructed reusable garbled Turing machines by employing techniques of FHE, witness encryption (WE) schemes, and the existence of SNARKs (Succinct Non-interactive Arguments of Knowledge). It would be interesting to know whether one can design reusable garbled Turing machines without using FHE schemes.

Using Garg et al's indistinguishability obfuscators for NC^1 [12], this paper designs a Turing machine garbling scheme without using FHE schemes. Though it is not clear whether multilinear maps based indistinguishability obfuscators could be more efficient than FHE, the goal of this paper is to develop alternative techniques for resuable garbling schemes to stimulate further research in this direction. The techniques that we used to construct garbled Turing machines could also be used to construct indistinguishability obfuscators for all polynomial size circuits without FHE schemes. Though it has been shown (see, e.g., [1]) that several proposed cryptographic multilinear map construction techniques are insecure, there has been a promising trend (see, e.g., Huang [18,19]) of using Weil descent to design secure trilinear maps. If the security of these Weil descent based trilimear maps could be verified, they should be sufficient for our garbled Turing machine design.

Independently of this work, Koppula, Lewko, and Waters [21] recently designed indistinguishability obfuscation for Turing machines using Garg et al's indistinguishability obfuscators, one-way functions and injective pseudo random generators. Some other recent related works on iterated circuit based garbling schemes could be found in Lin and Pass [22], Bitansky, Garg, and Telang [6], Canetti and Holmgren [10], Garg, Lu, Ostrovsky, and Scafuro [14], and Cannetti, Holmgren, Jain, and Vaikuntanathan [9]. It should also be noted that Boyle, Chung, and Pass [7] and Ananth, Boneh, Garg, Sahai, and Zhandry [2] showed how to transform Garg et al's indistinguishability obfuscators into one that operates on Turing machines with a strong security assumption called differing input obfuscation.

We conclude this section with the introduction of some notations. A Turing machine is defined as a 5-tuple $M = \langle Q, \Gamma, \delta, q_0, q_F \rangle$ with the properties:

- Q is a finite, non-empty set of states.
- Γ is a finite, non-empty set of tape alphabet symbols. Among symbols in Γ, a special symbol $B \in \Gamma$ is the blank symbol.
- $q_0 \in Q$ is the initial state, and $q_F \in Q$ is the final accepting state.
- $\delta : (Q \setminus \{q_F\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, where L is left shift, R is right shift.

A Turing machine M is called oblivious (OTM) if there exists a function $s(t)$ such that M's head is at cell position $s(t)$ at time t regardless of M's input values. Since every $T(n)$-time bounded Turing Machine can be simulated by an $O(T(n)\log(T(n)))$-time bounded OTM (see, Pippenger and Fischer [23]) all along this paper all TMs are oblivious.

For our garbled Turing machine $\overline{M}$, it takes approximately the same time for $\overline{M}$ to stop on an encrypted input $\bar{x}$ as that the un-garbled M to stop on the un-encrypted input x. If the running time of Turing machines on specific inputs needs to be protected, then one can easily modify Turing machines in such a way that it takes the same time to stop on all inputs of the same length. The details are omitted in this paper.

For a string $x \in \Gamma^*$, we use $x[i]$ to denote the ith element of x. That is, $x = x[0] \cdots x[n-1]$ where n is the length of x. We use $x \in_R \Gamma$ to denote that x is randomly chosen from Γ with the uniform distribution. We use κ to denote the security parameter, $p(\cdot)$ to denote a function p that takes one input, and $p(\cdot,\cdot)$ to denote a function p that takes two inputs. A function f is said to be negligible in an input parameter κ if for all $d > 0$, there exists n_0 such that for all $\kappa > n_0$, $f(\kappa) < \kappa^{-d}$. For convenience, we write $f(\kappa) = \mathrm{negl}(\kappa)$. Two ensembles, $X = \{X_\kappa\}_{\kappa \in N}$ and $Y = \{Y_\kappa\}_{\kappa \in N}$, are said to be computationally indistinguishable if for all probabilistic polynomial-time algorithm D, we have

$$|\mathtt{Prob}[D(X_\kappa, 1^\kappa) = 1] - \mathtt{Prob}[D(Y_\kappa, 1^\kappa) = 1]| = \mathrm{negl}(\kappa).$$

Throughout the paper, we use probabilistic experiments and denote their outputs using random variables. For example, $\mathrm{Exp}^{\mathrm{real}}_{E,A}(1^\kappa)$ represents the output of the real experiment for scheme E with adversary A on security parameter κ. Throughout the paper, $\mathtt{E} = (\mathtt{E.KeyGen}, \mathtt{E.Enc}, \mathtt{E.Dec})$ denotes a semantically secure symmetric-key encryption scheme and $\mathtt{PK} = (\mathtt{PK.KeyGen}, \mathtt{PK.Enc}, \mathtt{PK.Dec})$ denotes a semantically secure public-key encryption scheme.

The structure of this paper is as follows. In Section 2, we briefly present the intuition for our construction of reusable garbled Turing machines. Section 3 reviews the reusable circuit garbling scheme for NC^1 circuits. Section 4 presents the construction of reusable garbled Turing machines with $\mathtt{ABE}_2$. Section 5 presents the construction of reusable garbled Turing machines without $\mathtt{ABE}_2$.

2 Overview of our construction

In this section, we describe the intuition underlying our constructions. Assuming the hardness of multilinear Jigsaw puzzles (that is, in the generic multilinear encoding model), Garg et al [12] constructed functional encryption schemes for NC^1 circuits with succinct ciphertexts. Garg et al then extended their results to all polynomial size circuits using fully homomorphic encryption (FHE) schemes. As a corollary, for each circuit $C \in NC^1$, one can construct a reusable garbled circuit $\overline{C}$ without using FHE schemes. After Garg et al's [12] work, several other obfuscators for complexity class NC^1 have been proposed. For example, Brakerski and Rothblum [8] and Barak et al [3] designed virtual black-box obfuscators for NC^1 without using FHE.

In the following, we present our idea of constructing reusable Turing machines garbling schemes without using FHE schemes. This construction can also be used to design succinct ciphertext functional encryption schemes for all Turing machines without employing FHE schemes. Based on Pippenger and Fischer's results [23], we assume that the given Turing machine M is oblivious. In order to garble a Turing machine M, the transition function δ of M is converted to a circuit C_δ.

The circuit C_δ takes three inputs: the current head tape symbol b, the current state q of M, and a special session control tape that contains the session information and seed for pseudorandom generators. This session control tape is used to provide session information for C_δ to check whether the current head tape symbol and the current state of M are consistent. There are several reasons for including this additional tape. For example, it could be used to prevent the adversary from feeding tape symbols and Turing machine states from one execution of $\overline{M}(x)$ to another execution of $\overline{M}(y)$ and to prevent the adversary from replaying the execution of $\overline{M}(x)$ on an early or later stage of the tape cell contents. Given these inputs, C_δ checks whether these inputs are consistent (e.g., all of them contain the same random session identification string, the counters are consistent, and the tape cell is the most recently updated one). If the inputs are consistent, C_δ finds the matching transition rule and outputs the next head state q' and changes the current tape symbol to b'. Since C_δ will be converted to Garg et al's reusable garbled circuit $\overline{C}_\delta$ and $\overline{C}_\delta$ only accepts appropriately encoded inputs, both q' and b' need to be appropriately encoded by C_δ before output. Using information from the three inputs, C_δ updates the session control tape and uses the encoding key (that is, the public key of a public key encryption scheme, this key could either be included as part of the input to the circuit C_δ or be built-in C_δ itself) for Garg et al's NC^1 circuit garbling scheme to encode the session control tape, the tape symbol b', and the state q' respectively. The encoding process consists of encrypting the corresponding values using two public keys at the same time and constructing a statistically simulation sound NIZK proof that the two cipher texts are the encryption of the same plain text[3]. Furthermore, if the output state is q_F, C_δ may output the Turing machine state q_F in clear text without encoding so that the evaluator knows that the Turing machine M stops. Gentry et al [15] showed that it is possible to check whether a memory cell value is the most recently updated one using NC^1 circuits, and Ishai et al [20] showed that general cryptographic primitives such as encryptions and commitments could be constructed in NC^1 assuming the existence of NC^0 pseudorandom generators. Thus C_δ could be easily constructed in NC^1 with reasonable assumptions. Indeed, Garg et al's construction [12] requires the existence of an NC^1 decryption circuit for a public key encryption scheme. In a summary, circuit C_δ could be constructed in NC^1. Thus Garg et al's approach [12] implies a reusable garbled circuit $\overline{C}_\delta$ for C_δ without FHE.

Depending on the application scenario, the evaluator may or may not need to decrypt the encrypted final output of the Turing machine execution $M(x)$. For example, if a client submits the garbled Turing machine to a cloud data server to carry out computation on his encrypted data at the cloud, the cloud only needs to return the encrypted output to the client without decryption. However, in functional encryption schemes or other applications, the evaluator needs to learn the decrypted output $M(x)$. In this case, the innovative ideas by Goldwasser et al [16] can be used to decrypt the output. That is, an $\mathtt{ABE}_2$ scheme for Turing machines is used to provide keys for Yao's one-time garbled circuit to decrypt the output. The details are presented in Section 4.

[3] Note that if we use recent virtual black-box obfuscators by Brakerski and Rothblum [8] and Barak et al [3], then it is sufficient to encode the input using one public key and no NIZK proof is needed.

Attribute-Based Encryption scheme $\mathtt{ABE}_2$ for Turing machines are relatively slow. In order to improve the efficiency, we can design another reusable garbled circuit to decrypt the encrypted output $\overline{M(x)}$. For this approach, special caution needs to be taken. For example, an active adversary may manipulate the encoded input tape $\bar{x}$ by swapping/repeating tape cells for $\bar{x}$ to obtain a valid encoded tape for $x' \neq x$. The adversary may then run $\overline{M}$ on $\bar{x}'$ to obtain $\overline{M}(x')$ and run the reusable decryption circuit to decrypt $\overline{M}(x')$. In order to address these challenges, we use Chaitin's universal self-delimiting Turing machines [11]. The input to a self-delimiting Turing machine must be encoded in a prefix-free domain. Without proper encoded prefix-free input, the self-delimiting Turing machine would not enter the q_F state. secure message authentication tags for inputs. Furthermore, we also revise the Turing machine in such a way that before entering the state q_F, it selects a random secret key $\mathtt{sk}_\mathtt{o}$ and encrypts the output $M(x)$. That is, the output tape contains encoded $\mathtt{sk}_\mathtt{o}||\mathtt{E.Enc}(\mathtt{sk}_\mathtt{o}, M(x))$. An NC^1 circuit C_d is constructed to decrypt the output tape to $\mathtt{sk}_\mathtt{o}||\mathtt{E.Enc}(\mathtt{sk}_\mathtt{o}, M(x))$ first and then use $\mathtt{sk}_\mathtt{o}$ to decrypt the actual output $M(x)$. Since such kind of circuits exist in NC^1, Garg et al's approach [12] can be used to obtain a reusable garbled circuit $\overline{C}_d$. The detailed construction of C_d is presented in Section 5.

In a summary, for each evaluation of a Turing machine M on an input x, Turing machine owner pads each bit $x[i]$ with appropriate session control information to $x[i]||\mathtt{session}_b||i||0$ and uses encoding keys for Garg's NC^1 reusable garbling scheme to encode it to $\mathtt{RGC}_{\mathtt{nc}^1}.\mathtt{Enc}(\mathtt{gsk}, x[i]||\mathtt{session}_b||i||0)$ as the content of the i-th tape cell. The padded suffix 0 denotes that this cell value is an input value. When a cell value is modified at Turing machine step j, the value j is placed in the suffix. The Turing machine owner also encodes the session control information (e.g., `session` and random seeds) and provides them in the session control tape and encodes the Turing machine initial state q_0. The evaluator uses $\overline{C}_\delta$ to simulate the Turing machine M and uses an $\mathtt{ABE}_2$ scheme or another reusable garbled circuit to decrypt the output.

3 Reusable garbled circuits for NC^1

In this section, we review necessary techniques that are required for our construction. We first present the formal definition of one-time and reusable garbling schemes for circuits and Turing machines.

Definition 1. *Let $\mathcal{M} = \{\mathcal{M}_n\}_{n\in N}$ be a family of circuits/Turing machines such that $\mathcal{M}_n$ is a set of functions that take n-bit inputs. A garbling scheme for $\mathcal{M}$ is a tuple of probabilistic polynomial time algorithms $\mathtt{GS} = (\mathtt{GS.Garble}, \mathtt{GS.Enc}, \mathtt{GS.Eval})$ with the following properties*

- $(\overline{M}, \mathtt{gsk}) = \mathtt{GS.Garble}(1^\kappa, M)$ *outputs a garbled circuit/Turing machine $\overline{M}$ and a secret key $\mathtt{gsk}$ for $M \in \mathcal{M}_n$ on the security parameter input κ.*
- $c_x = \mathtt{GS.Enc}(\mathtt{gsk}, x)$ *outputs an encoding c_x for an input $x \in \{0,1\}^*$.*
- $y = \mathtt{GS.Eval}(\overline{M}, c_x)$ *outputs a value y which should equal to $M(x)$.*

The garbling scheme $\mathtt{GS}$ *is* correct *if the probability that $\mathtt{GS.Eval}(\overline{M}, c_x) \neq M(x)$ is negligible. The garbling scheme* $\mathtt{GS}$ *is* efficient *if the size of $\overline{M}$ is bounded by a polynomial and the run-time of $c_x = \mathtt{GS.Enc}(\mathtt{gsk}, x)$ is also bounded by a polynomial.*

Throughout this paper, we will use $\mathtt{GT} = (\mathtt{GT.Garble}, \mathtt{GT.Enc}, \mathtt{GT.Eval})$ and $\mathtt{GC} = (\mathtt{GC.Garble}, \mathtt{GC.Enc}, \mathtt{GC.Eval})$ to denote a garbling scheme for Turing machines and a garbling scheme for circuits respectively. Similarly, we will use $\mathtt{RGT} = (\mathtt{RGT.Garble}, \mathtt{RGT.Enc}, \mathtt{RGT.Eval})$ and $\mathtt{RGC} = (\mathtt{RGC.Garble}, \mathtt{RGC.Enc}, \mathtt{RGC.Eval})$ to denote reusable garbling schemes for Turing machines and circuits respectively.

The security of garbling schemes is defined in terms of input and circuit privacy in the literature. The following security definition for one-time garbling schemes based on Bellare, Hoand, and Rogaway [4] captures the intuition that for any circuit or input chosen by the adversary, one can simulate the garbled circuit and the encoding based on the computation result in polynomial time. In the definition, the variable α represents any state that A may want to give to D.

Definition 2. *(Input and circuit privacy for one-time garbling schemes) A garbling scheme* $\mathtt{GS}$ *for a family of circuits/Turing machine* $\mathcal{M}$ *is said to be* input and circuit private *if there exists a probabilistic polynomial time simulator* $\mathtt{Sim}_{\mathtt{GS}}$ *such that for all probabilistic polynomial time adversaries* A *and* D *and all large* κ*, we have*

$$\left|\mathtt{Prob}[D(\alpha, x, M, \overline{M}, c) = 1 | \mathtt{REAL}] - \mathtt{Prob}[D(\alpha, x, M, \tilde{M}, \tilde{c}) = 1 | \mathtt{SIM}]\right| = \mathrm{negl}(\kappa)$$

where $\mathtt{REAL}$ *and* $\mathtt{SIM}$ *are the following events*

$$\begin{array}{ll}
\mathtt{REAL}: & \mathtt{SIM}: \\
(x, M, \alpha) \leftarrow A(1^\kappa) & (x, M, \alpha) \leftarrow A(1^\kappa) \\
(\overline{M}, \mathtt{gsk}) \leftarrow \mathtt{GS.Garble}(1^\kappa, M) & (\tilde{M}, \tilde{c}_x) \leftarrow \mathtt{Sim}_{\mathtt{GS}}(M(x), 1^{\max\{\kappa, |M|, |x|\}}) \\
c_x \leftarrow \mathtt{GS.Enc}(\mathtt{gsk}, x) &
\end{array}$$

The privacy for reusable garbling schemes is defined also in terms of circuit and input privacy and the reader is referred to Goldwasser et al [17] for details.

Definition 3. *(Private reusable garbling schemes, adapted from Goldwasser et al [17]) Let* $\mathtt{RGS}$ *be a reusable garbling scheme for a family of Turing machines/circuits* $\mathcal{M} = \{\mathcal{M}_n\}_{n \in N}$ *and* $M \in \mathcal{M}_n$ *be a Turing machine/circuit with* n*-bits inputs. For a pair of probabilistic polynomial time algorithms* $A = (A_0, A_1)$ *and a probabilistic polynomial time simulator* $S = (S_0, S_1)$*, define two experiments:*

$$\begin{array}{ll}
\mathtt{Exp}^{\mathtt{real}}_{\mathtt{RGS},A}(1^\kappa): & \mathtt{Exp}^{\mathtt{ideal}}_{\mathtt{RGS},A,S}(1^\kappa): \\
(M, \mathtt{state}_A) = A_0(1^\kappa) & (M, \mathtt{state}_A) = A_0(1^\kappa) \\
(\mathtt{sk}, \overline{M}) = \mathtt{RGS.Garble}(1^\kappa, M) & (\tilde{M}, \mathtt{state}_S) = S_0(1^\kappa, M) \\
\alpha = A_1^{\mathtt{RGS.Enc}(\mathtt{sk}, \cdot)}(M, \overline{M}, \mathtt{state}_A) & \alpha = A_1^{O(\cdot, M)[[\mathtt{state}_S]]}(M, \tilde{M}, \mathtt{state}_A)
\end{array}$$

In the above experiments, $O(\cdot, M)[[\mathtt{state}_S]]$ *is an oracle that on input* x *from* A_1*, runs* S_1 *with inputs* $1^{|x|}$*,* $M(x)$*, and the latest state of* S*; it returns the output of* S_1 *(storing the new simulator state for the next invocation). The garbling scheme* $\mathtt{RGS}$ *is said to be* private *with reusability if there exists a probabilistic polynomial time simulator* S *such that for all pairs of probabilistic polynomial time adversaries* $A = (A_0, A_1)$*, the following two distributions are computationally indistinguishable:*

$$\left\{\mathtt{Exp}^{\mathtt{real}}_{\mathtt{RGS},A}(1^\kappa)\right\}_{\kappa \in N} \approx_c \left\{\mathtt{Exp}^{\mathtt{ideal}}_{\mathtt{RGS},A,S}(1^\kappa)\right\}_{\kappa \in N} \tag{1}$$

The recent virtual black-box obfuscators for NC^1 by Brakerski-Rothblum [8] and Barak et al [3] require generic multilinear encoding model. Though Garg et al's [12] indistinguishability obfuscator for NC^1 is constructed using generic multilinear encoding also, it does not rule out the possibility of constructing indistinguishability obfuscators in the plain model with weaker assumptions. Garg et al [12] showed the following theorem.

Theorem 1. *Assuming the existence of an indistinguishability obfuscator, there is a garbling scheme* $\mathtt{RGC}_{\mathtt{nc}^1}$ *for* NC^1 *circuits that is secure according to the definition in Goldwasser et al [17].*

Based on witness encryption (WE) schemes by Garg et al [13] and the existence of SNARKs (Succinct Non-interactive Arguments of Knowledge) by Bitansky et al [5], Goldwasser et al [16] designed attribute-based encryption (ABE) schemes for Turing machines[4]. The single-outcome ABE schemes for Turing machines in [16] could be converted to two-outcome attribute-based encryption schemes ($\mathtt{ABE}_2$) for Turing machines using the techniques from Goldwasser et al [17].

Goldwasser et al [17] introduced the following concept of two-outcome attribute-based encryption schemes ($\mathtt{ABE}_2$) for Turing machines.

Definition 4. *A two-outcome attribute-based encryption scheme* $\mathtt{ABE}_2$ *for a class of Turing machines* $\mathcal{M}$ *is a tuple of four algorithms* ($\mathtt{ABE}_2.\mathtt{Setup}$, $\mathtt{ABE}_2.\mathtt{Enc}$, $\mathtt{ABE}_2.\mathtt{KeyGen}$, $\mathtt{ABE}_2.\mathtt{Dec}$)*:*

- $(\mathtt{mpk}, \mathtt{msk}) = \mathtt{ABE}_2.\mathtt{Setup}(1^\kappa)$*: On the security parameter input* 1^κ*, outputs the master public key* $\mathtt{mpk}$ *and the master secret key* $\mathtt{msk}$*.*
- $\mathtt{sk}_M = \mathtt{ABE}_2.\mathtt{KeyGen}(\mathtt{msk}, M)$*: On input* $\mathtt{msk}$ *and a Turing machine* M*, outputs a secret key* $\mathtt{sk}_M$ *corresponding to* M*. Note that* M *is public.*
- $c = \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}, x, b_0, b_1)$*: On input the master public key* $\mathtt{mpk}$*, an attribute* $x \in \{0,1\}^*$*, and two messages* b_0, b_1*, outputs a ciphertext* c*.*
- $b_i = \mathtt{ABE}_2.\mathtt{Dec}(\mathtt{sk}_M, c)$*: On input a secret key* $\mathtt{sk}_M$ *for the Turing machine* M *and a ciphertext* c*, outputs* b_i *if* $M(x) = i$ *for* $i = 0, 1$*.*

Correctness (informal). *The correctness of an* $\mathtt{ABE}_2$ *scheme means* $\mathtt{ABE}_2.\mathtt{Dec}(\mathtt{sk}_M, c)$ *fails with a negligible probability (for a formal definition, it is referred to [17]).*

The security of an $\mathtt{ABE}_2$ scheme means that if one has the secret key $\mathtt{sk}_M$ for a Turing machine M, then one can decrypt one of the two encrypted messages based on the value of $M(x)$ where x is the attribute, but learns zero information about the other message. The formal definition could be found in Goldwasser et al [17].

Formally, the security can be defined as follows.

Definition 5. *(Goldwasser et al [17]) Let* $\mathtt{ABE}_2$ *be a two-outcome attribute-based encryption scheme for a class of Turing machines* $\mathcal{M}$*. Let* $A = (A_1, A_2, A_3)$ *be a tuple of probabilistic polynomial time adversaries. Define the experiment* $\mathtt{Exp}_{\mathtt{ABE}_2}(1^\kappa)$*:*

1. $(\mathtt{mpk}, \mathtt{msk}) = \mathtt{ABE}_2.\mathtt{Setup}(1^\kappa)$
2. $(M, \mathtt{state}_1) = A_1(\mathtt{mpk})$
3. $\mathtt{sk}_M = \mathtt{ABE}_2.\mathtt{KeyGen}(\mathtt{msk}, M)$

[4] Note that the **correctness** definition of Definition 3 for ABE in [16] is messed up.

4. $(a, a_0, a_1, x, \mathtt{state}_2) = A_2(\mathtt{state}_1, \mathtt{sk}_M)$ *where* a, a_0, a_1 *are bits*
5. *Choose a random bit* b *and let*

$$c = \begin{cases} \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}, x, a, a_b), & \text{if } M(x) = 0, \\ \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}, x, a_b, a), & \text{otherwise.} \end{cases}$$

6. $b' = A_3(\mathtt{state}_2, c)$. *If* $b = b'$*, then output* 1*, else output* 0.

The scheme is said to be a single-key fully-secure two-outcome $\mathtt{ABE}_2$ *if for all probabilistic polynomial time adversaries* A *and for all sufficiently large security parameters* κ*, we have*

$$\mathtt{Prob}[\mathtt{Exp}_{\mathtt{ABE}_2, A}(1^\kappa) = 1] \leq 1/2 + \mathrm{negl}(\kappa).$$

The scheme is said to be single-key selectively secure if A *needs to provide* x *before receiving* $\mathtt{mpk}$.

4 Reusable garbled Turing machines with $\mathtt{ABE}_2$

The construction of a garbling scheme $\mathtt{RGT} = (\mathtt{RGT.Garble}, \mathtt{RGT.Enc}, \mathtt{RGT.Eval})$ for Turing machines M proceeds as follows.

$(\mathtt{gsk}, \overline{M}) = \mathtt{RGT.Garble}(1^\kappa, M)$:

- $\mathtt{sk} = \mathtt{E.KeyGen}(1^\kappa)$, $(\mathtt{psk}_i, \mathtt{ppk}_i) = \mathtt{PK.KeyGen}(1^\kappa)$ for $i = 0, 1$.
- Let $E_M = \mathtt{E.Enc}(\mathtt{sk}, M)$ and $\overline{\mathtt{sk}} = \mathtt{PK.Enc}(\mathtt{ppk}_0, \mathtt{sk})$.
- Let U_M be an oblivious universal Turing machine and let $s(t)$ be the head position function for U_M. On input x, U_M first decrypts $\mathtt{sk} = \mathtt{PK.Dec}(\mathtt{psk}_0, \overline{\mathtt{sk}})$ and $M = \mathtt{E.Dec}(\mathtt{sk}, E_M)$. U_M then runs M on x to output $M(x)$.
- Let δ be the transition function of U_M and $C_\delta \in NC^1$ be the following circuit:

Input: head sate $q||\texttt{session}_q||j_q$, tape cell $b||\texttt{session}_b||j_c||j_b$, and control tape $\texttt{ctape} = \texttt{ppk}_0||\texttt{ppk}_1||\texttt{state}||\texttt{session}_s||j_q$.

1. use information from $\texttt{session}_s$ to extract the current Turing machine step j_1, expected current head position $j_2 = s(j_1)$, and the most recent time j_3 that the cell j_c has been updated.
2. if $j_1 \neq j_q + 1$ or $j_2 \neq j_c$ or $j_3 \neq j_b$, go to step 11.
3. if $\texttt{session}_s, \texttt{session}_b$, and $\texttt{session}_q$ are inconsistent, go to step 11.
4. if $q = q_F$, output the state q_F in clear and exit.
5. if $q = q_{\texttt{noop}}$, go to step 11.
6. compute the next state and tape symbols $(q', b') = \delta(q, b)$.
7. update ideal cipher $\texttt{E}$ state, public key cipher $\texttt{PK}$ state, and the values in $\texttt{session}_s, \texttt{session}_b$, $\texttt{session}_q$, and $\texttt{state}$.
8. let $e_i^{q'} = \texttt{PK.Enc}(\texttt{ppk}_i, q'||\texttt{session}_q||j_1)$ for $i = 0, 1$, and $\pi^{q'}$ be a statistically simulation sound non-interactive zero knowledge (NIZK) proof for the following $\texttt{NP}$ statement: $e_0^{q'}$ and $e_1^{q'}$ are encryptions of a same message using public keys $\texttt{ppk}_0$ and $\texttt{ppk}_1$.
9. Similarly, compute $(e_0^{b'}, e_1^{b'}, \pi^{b'})$ for tape cell $b'||\texttt{session}_b||j_c||j_1$ and $(e_0^{\texttt{ctape}}, e_1^{\texttt{ctape}}, \pi^{\texttt{ctape}})$ for $\texttt{ctape} = \texttt{ppk}_0||\texttt{ppk}_1||\texttt{state}||\texttt{session}_s||j_1$
10. write $(e_0^{b'}, e_1^{b'}, \pi^{b'})$ to tape cell, output next state $(e_0^{q'}, e_1^{q'}, \pi^{q'})$, and update control tape as $(e_0^{\texttt{ctape}}, e_1^{\texttt{ctape}}, \pi^{\texttt{ctape}})$. Exit.
11. let $q' = q_{\texttt{noop}}$, $b' = 0$, and go to step 8.

– Let $\overline{C}_\delta = \texttt{RGC}_{\texttt{nc}^1}.\texttt{Garble}(1^\kappa, \texttt{ppk}_0, \texttt{ppk}_1, \texttt{psk}_0, C_\delta)$. Here we provide the parameters $\texttt{ppk}_0, \texttt{ppk}_1$, and $\texttt{psk}_0$ to $\texttt{RGC}_{\texttt{nc}^1}.\texttt{Garble}$ to overwrite the corresponding internal key generation process within $\texttt{RGC}_{\texttt{nc}^1}.\texttt{Garble}$.
– Let $\overline{U}_M = (s(t), \overline{C}_\delta)$ be a Turing machine that uses $\overline{C}_\delta$ to simulate the transition function δ of U_M.
– Let $\omega = \omega(\kappa)$ be the length of the total garbled outputs in the $\overline{U}_M$ under the security parameter κ.
– Run $\texttt{ABE}_2.\texttt{Setup}(1^\kappa)$ algorithm ω times: $(\texttt{mpk}_i, \texttt{msk}_i) \leftarrow \texttt{ABE}_2.\texttt{Setup}(1^\kappa)$ for $i < \omega$ and let

$$\texttt{msk} = (\texttt{msk}_0, \cdots, \texttt{msk}_{\omega-1}) \text{ and } \texttt{mpk} = (\texttt{mpk}_0, \cdots, \texttt{mpk}_{\omega-1}).$$

– Let $\overline{U}_M^i(\cdot)$ be the ith bit of the output of running $\overline{U}_M$ on an encoded input.
– Run $\texttt{ABE}_2.\texttt{KeyGen}(\texttt{msk}, \cdot)$ for each of the function $\overline{U}_M^i(\cdot)$ under the different master secret keys to construct secret keys:

$$\texttt{gM}_i \leftarrow \texttt{ABE}_2.\texttt{KeyGen}(\texttt{msk}_i, \overline{U}_M^i(\cdot)) \text{ for } i < \omega.$$

– Output $\overline{M} = (\texttt{gM}_0, \cdots, \texttt{gM}_{\omega-1})$ and $\texttt{gsk} = (\texttt{ppk}_0, \texttt{ppk}_1, \texttt{psk}_0, \texttt{mpk})$.

$c_x = \texttt{RGT.Enc}(\texttt{gsk}, x)$:

– Generate $\texttt{state}$ uniformly at random for the input string x.
– Update session identification values $\texttt{session}_s, \texttt{session}_b, \texttt{session}_q$.
– For each input tape cell j, let $e_i^j = \texttt{PK.Enc}(\texttt{ppk}_i, x[j]||\texttt{session}_b||j||0)$ for $i = 0, 1$, and π^j be a statistically simulation sound non-interactive zero knowledge (NIZK)

proof for the following NP statement: e_0^j and e_1^j are encryptions of a same message using public keys $\mathtt{ppk}_0$ and $\mathtt{ppk}_1$.

- Similarly, compute $(e_0^{q_0}, e_1^{q_0}, \pi^{q_0})$ for the initial head state $q_0||\mathtt{session}_q||0$ and $(e_0^{\mathtt{ctape}}, e_1^{\mathtt{ctape}}, \pi^{\mathtt{ctape}})$ for $\mathtt{ctape} = \mathtt{ppk}_0||\mathtt{ppk}_1||\mathtt{state}||\mathtt{session}_s||0$.
- Let $c = \left\{(e_0^{q_0}, e_1^{q_0}, \pi^{q_0}), (e_0^{\mathtt{ctape}}, e_1^{\mathtt{ctape}}, \pi^{\mathtt{ctape}}), (e_0^j, e_1^j, \pi^j) : 0 \leq j \leq n-1\right\}$.
- Let $C_d \in NC^1$ be the following circuit:

> **Input:** encoded output tape $\overline{\mathtt{otape}}$ and encoded control tape $\overline{\mathtt{ctape}}$.
> 1. decrypt output tape $(\mathtt{otape}, \mathtt{session}_b) = \mathtt{PK.Dec}(\mathtt{psk}_0, \overline{\mathtt{otape}})$ and current control tape $(\mathtt{state}, \mathtt{session}_s) = \mathtt{PK.Dec}(\mathtt{psk}_0, \overline{\mathtt{ctape}})$.
> 2. if $\mathtt{session}_s$ and $\mathtt{session}_b$ are inconsistent, exit.
> 3. write otape to output tape and exit.

- Run Yao's one-time garbled circuit generation algorithm to produce a garbled circuit Λ: $\{0,1\}^\omega \to \{0,1\}$ together with 2ω labels L_i^b for $i < \omega$ and $b \in \{0,1\}$.

$$\left(\Lambda, \{L_i^0, L_i^1\}_{i=0}^{\omega-1}\right) = \mathtt{GC.Garble}(1^\kappa, C_d).$$

- Produce $\mathtt{ABE}_2$ ciphertexts $c_0, \cdots, c_{\omega-1}$ as follows:

$$c_i \leftarrow \mathtt{ABE}_2\mathtt{.Enc}(\mathtt{mpk}_i, c, L_i^0, L_i^1) \text{ for } i < \omega.$$

- Output the cipher texts $c_x = (\Lambda, c_0, \cdots, c_{\omega-1})$.

$M(x) = \mathtt{RGT.Eval}(\overline{M}, c_x)$:

- Run $\mathtt{ABE}_2$ decryption algorithm on ciphertexts $c_0, \cdots, c_{\omega-1}$ to calculate the labels for Yao's garbled circuit Λ for $d_i = \overline{M}^i(c_x)$:

$$L_i^{d_i} \leftarrow \mathtt{ABE}_2\mathtt{.Dec}(\mathtt{gM}_i, c_i) \text{ for } i < \omega$$

- Evaluate the garbled circuit Λ with labels $L_i^{d_i}$ to compute the output $M(x)$

$$M(x) = \mathtt{GC.Eval}(\Lambda, L_i^{d_0}, \cdots, L_{\omega-1}^{d_{\omega-1}})$$

Proof of security

The correctness and efficiency of the reusable Turing machine garbling scheme RGT in the preceeding paragraph is straightforward. In the following, we show that the scheme RGT is private with reusability according to the definition in Goldwasser et al [17].

Assume that a Turing machine M is selected with the security parameter κ. We need to construct a simulator $S = (S_0, S_1)$ such that (1) holds for the reusable garbled Turing machine $\overline{M} = (\mathtt{gM}_0, \cdots, \mathtt{gM}_{\omega-1})$, assuming that there are a simulator $S_\delta = (S_{\delta,0}, S_{\delta,1})$ satisfying the security definition in Goldwasser et al [17]. reusable garbled circuits $\overline{C}_\delta$ and a simulator $\mathtt{Sim}_{\mathtt{GS}}$ satisfying Definition 2 for Yao's one-time garbling scheme.

To generate a simulated garbled Turing machine $\tilde{M} = (\tilde{\mathtt{gM}}_0, \cdots, \tilde{\mathtt{gM}}_{\omega-1})$ for the Turing machine M, S_0 runs the following procedures:

1. Generate fresh mpk and msk as in RGT.Garble process.
2. Run simulators S_δ to generate a reusable garbled circuit $\tilde{C}_\delta$.

3. Run $\mathtt{ABE}_2.\mathtt{KeyGen}(\mathtt{msk}, \cdot)$ to generate $\tilde{M} = (\tilde{\mathtt{gM}}_0, \cdots, \tilde{\mathtt{gM}}_{\omega-1})$.

During the simulation, S_1 receives the latest simulator's state, $1^{|x|}$, $\tilde{C}_\delta$, and a Turing machine output $M(x)$ for some input x without seeing the value of x. S_1 needs to output a simulated encoding $\tilde{c} = (\tilde{\Lambda}, \tilde{c}_0, \cdots, \tilde{c}_{\omega-1})$ for the $\mathtt{RGT.Eval}$ process without access to C_d. Let $\mathtt{Sim}_{\mathtt{GS}}$ be the simulator from Definition 2 for Yao's one-time garbling scheme. Run $\mathtt{Sim}_{\mathtt{GS}}$ to produce a simulated garbled circuit $\tilde{\Lambda}$ for the circuit C_d together with the simulated encoding consisting of ω labels $\tilde{L}_i$ for $i = 0, \cdots, \omega - 1$. That is, we have

$$\left(\tilde{\Lambda}, \tilde{L}_0, \cdots, \tilde{L}_{\omega-1}\right) = \mathtt{Sim}_{\mathtt{GS}}(1^\kappa, M(x), 1^\omega).$$

S_1 can invoke the above simulation since it knows $M(x)$ and the size of input to C_d (that is, the output size of $\tilde{C}_\delta$). S_1 can then produce the simulated $\mathtt{ABE}_2$ ciphertexts $\tilde{c}_0, \cdots, \tilde{c}_{\omega-1}$ as follows:

$$\tilde{c}_i \leftarrow \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}_i, \tilde{c}_x, \tilde{L}_i, \tilde{L}_i) \text{ for } i < \omega.$$

Note that we used the label $\tilde{L}_i$ for two times. In a summary, S_1 can now output the simulated encoding $(\tilde{\Lambda}, \tilde{c}_0, \cdots, \tilde{c}_{\omega-1})$.

Now it suffices to show that the simulation satisfies the security definition in Goldwasser et al [17]. for any adversary $A = (A_0, A_1)$. Without loss of generality, we may assume that A_1 output α equals to its entire view. That is, all information that A_1 has received during the protocol run. Note that if we could prove that the real and ideal experiment outputs are computationally indistinguishable with this kind of output, it will be computationally indistinguishable with any other kind of outputs since A_1 is a probabilistic polynomial time algorithm. That is, any output should be probabilistic polynomial time computable from this view. In the following, we define five games first.

Game 0: The ideal game $\mathtt{Exp}^{\mathtt{ideal}}_{\mathtt{RGT},A,S}(1^\kappa)$ of the security definition in Goldwasser et al [17] with simulator S. The output distribution for this game is:

$$M, \mathtt{gsk}, \mathtt{state}_A, \mathtt{ABE}_2.\mathtt{KeyGen}(\mathtt{msk}, \cdot),$$
$$\left\{ x_i, \tilde{c}_{x_i}, \mathtt{Sim}_{\mathtt{Garble}}(1^\kappa, M(x_i), 1^\omega), \left\{ \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}_i, \tilde{c}_{x_i}, \tilde{L}_{i,j}, \tilde{L}_{i,j}) \right\}_{j=0}^{\omega-1} \right\}_{i=0}^{t-1}$$

Game 1: The same as Game 0 except that the Turing machine M is replaced with the reusable garbled circuit $\overline{C}_\delta$ and the circuit C_d. That is, the output distribution for this game is:

$$\overline{C}_\delta, C_d, \mathtt{gsk}, \mathtt{state}_A, \mathtt{ABE}_2.\mathtt{KeyGen}(\mathtt{msk}, \cdot),$$
$$\left\{ x_i, \tilde{c}_{x_i}, \mathtt{Sim}_{\mathtt{Garble}}(1^\kappa, M(x_i), 1^\omega), \left\{ \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}_i, \tilde{c}_{x_i}, \tilde{L}_{i,j}, \tilde{L}_{i,j}) \right\}_{j=0}^{\omega-1} \right\}_{i=0}^{t-1}$$

Game 2: The same as Game 1 except that the simulated input encoding $\tilde{c}_{x_i}$ is replaced with the actual encoding c_{x_i} of x_i by encoding x_i using $\mathtt{gsk}$. Note that we keep $\tilde{c}_{x_i}$ unchanged within the $\mathtt{ABE}_2.\mathtt{Enc}$ procedure. That is, the output distribution for this game is:

$$\overline{C}_\delta, C_d, \mathtt{gsk}, \mathtt{state}_A, \mathtt{ABE}_2.\mathtt{KeyGen}(\mathtt{msk}, \cdot),$$
$$\left\{ x_i, c_{x_i}, \mathtt{Sim}_{\mathtt{Garble}}(1^\kappa, M(x_i), 1^\omega), \left\{ \mathtt{ABE}_2.\mathtt{Enc}(\mathtt{mpk}_i, \tilde{c}_{x_i}, \tilde{L}_{i,j}, \tilde{L}_{i,j}) \right\}_{j=0}^{\omega-1} \right\}_{i=0}^{t-1}$$

Game 3: The same as Game 2 except the simulated garbled circuit $\tilde{\Lambda}$ is replaced with the real garbled circuit Λ: $\left(\Lambda, \{L_j^0, L_j^1\}_{j=0}^{\omega-1}\right) = \mathtt{GC.Garble}(1^\kappa, C_d)$. The output distribution for this game is:

$$\overline{C}_\delta, C_d, \mathtt{gsk}, \mathtt{state}_A, \mathtt{ABE_2.KeyGen}(\mathtt{msk}, \cdot),$$
$$\left\{ x_i, c_{x_i}, \mathtt{GC.Garble}(1^\kappa, C_d), \left\{ \mathtt{ABE_2.Enc}(\mathtt{mpk}_i, \tilde{c}_{x_i}, \tilde{L}_{i,j}, \tilde{L}_{i,j}) \right\}_{j=0}^{\omega-1} \right\}_{i=0}^{t-1}$$

Game 4: The same as Game 3 except that the $\mathtt{ABE_2.Enc}$ ciphertext is replaced with the real $\mathtt{ABE_2}$ ciphertext. In other words, this is the real experiment $\mathtt{Exp}_{\mathtt{RGT},A}^{\mathtt{real}}(1^\kappa)$ of the security definition in Goldwasser et al [17]. The output distribution for this game is:

$$\overline{C}_\delta, C_d, \mathtt{gsk}, \mathtt{state}_A, \mathtt{ABE_2.KeyGen}(\mathtt{msk}, \cdot),$$
$$\left\{ x_i, c_{x_i}, \mathtt{GC.Garble}(1^\kappa, C_d), \left\{ \mathtt{ABE_2.Enc}(\mathtt{mpk}_i, c_{x_i}, L_{i,j}^0, L_{i,j}^1) \right\}_{j=0}^{\omega-1} \right\}_{i=0}^{t-1}$$

We prove that the outputs of each pair of games are computationally indistinguishable in the following lemmas. Thus our reusable garbled circuits are circuit and input private with reusability.

Lemma 1. *Assume that $\overline{C}_\delta$ is a secure reusable garbled circuit for C_δ in the simulation-based security model. Then the outputs of Game 0 and Game 1 are computationally indistinguishable.*

Proof (sketch). The proof is by contradiction. Assume that the outputs of Game 0 and Game 1 could be distinguished by a probabilistic polynomial time algorithm D. Then one can use D to construct a probabilistic polynomial time distinguisher D_1 to show that Theorem 1 is not true. Details are omitted here. □

Lemma 2. *Assume that both ciphers* $\mathtt{E.Enc}$ *and* $\mathtt{PK.Enc}$ *are semantically secure. Then the outputs of Game 1 and Game 2 are computationally indistinguishable.*

Proof (sketch). Assume that there exist probabilistic polynomial time adversaries $A = (A_1, A_2)$ and a probabilistic polynomial time distinguisher D armed with A that distinguishes outputs of Game 1 and Game 2 with a non-negligible probability. Then one can use the standard hybrid argument to construct a probabilistic polynomial time distinguisher D_1 to distinguish the cipher texts c_{x_i} from the simulated cipher text $\tilde{c}_{x_i}$ with a non-negligible probability for some $i_0 = 0, \cdots, t-1$. This is a violation that both ciphers $\mathtt{E.Enc}$ and $\mathtt{PK.Enc}$ are semantically secure. □

Lemma 3. *Assume that the one-time garbling scheme is secure in the sense of Definition 2. Then the outputs of Game 2 and Game 3 are computationally indistinguishable.*

Proof (sketch). Assume that there exist probabilistic polynomial time adversaries $A = (A_1, A_2)$ and a probabilistic polynomial time distinguisher D armed with A that distinguishes outputs of Game 2 and Game 3 with a non-negligible probability. Then one can build a probabilistic polynomial time distinguisher D_1 to distinguish the outputs of the simulator $\mathtt{Sim}_{\mathtt{Garble}}$ and $\mathtt{GC.Garble}$. This contradicts Definition 2. The details for the construction of D_1 are omitted here. □

Lemma 4. *Assume that the* $\mathtt{ABE}_2$ *scheme is secure in the sense of Goldwasser et al [17]. Then the outputs of Game 3 and Game 4 are computationally indistinguishable.*

Proof (sketch). Assume that there exist probabilistic polynomial time adversaries $A = (A_1, A_2)$ and a probabilistic polynomial time distinguisher D armed with A that distinguishes outputs of Game 3 and Game 4 with a non-negligible probability. Then one can build a probabilistic polynomial time adversary $A = (A_1, A_2, A_3)$ such that

$$\mathtt{Prob}[\mathtt{Exp}_{\mathtt{ABE}_2,A}(1^\kappa) = 1] > 1/2 + \mathrm{negl}(\kappa).$$

This contradicts the security definition of $\mathtt{ABE}_2$. Details for the construction of A are omitted here. □

Theorem 2. *Assume that the one-time garbling scheme is secure, the* $\mathtt{ABE}_2$ *scheme is secure, both ciphers* $\mathtt{E.Enc}$ *and* $\mathtt{PK.Enc}$ *are semantically secure, and* $\mathtt{RGC}_{\mathtt{nc}^1}$ *is secure according to Definition 2. Then the reusable garbling scheme* $\mathtt{RGT}$ *in Section 4 is secure according to the security definition in Goldwasser et al [17].*

Proof. This follows from Lemmas 1, 2, 3, and 4. □

5 Reusable garbled Turing machines without $\mathtt{ABE}_2$

In Section 4, we used Yao's one-time garbled circuits for C_d and Attribute Based Encryption ($\mathtt{ABE}_2$) schemes for Turing machines to decrypt the garbled Turing machine output $\overline{U}_M(\overline{x})$. $\mathtt{ABE}_2$ cipher text is relatively expensive to construct and the size of Yao's one-time garbled circuit for C_d is too large to be included in each garbled input. Thus it is preferred to use a reusable garbled circuit to decrypt the output of $\overline{U}_M(\overline{x})$. As mentioned in early sections, the challenge to use a garbled version $\overline{C}_d$ directly is that the adversary may use $\overline{C}_d$ to calculate $M(x')$ for input x' whose encoding is not provided by the Turing machine owner. To address these challenges, we use secure message authentication tags.

Let $\mathtt{MAC} = (\mathtt{MAC.KeyGen}, \mathtt{MAC.Enc}, \mathtt{MAC.Veri})$ be a secure message authentication scheme. The Turing machine M is revised to a new Turing machine $M_{\mathtt{mac}}$ as follows. The input to $M_{\mathtt{mac}}$ is first authenticated using a $\mathtt{MAC}$ scheme key and then encrypted using the semantically secure ideal cipher. That is, $M_{\mathtt{mac}}$ takes an input in format of $\bar{x} = \mathtt{E.Enc}(\mathtt{sk}_{\mathtt{mac}}, x||\mathtt{tag})$ where x is the supposed input to M and $\mathtt{tag} = \mathtt{MAC.Enc}(\mathtt{ask}, x)$. On an input $\bar{x}$, $M_{\mathtt{mac}}$ uses the built-in key $\mathtt{sk}_{\mathtt{mac}}$ to decrypt $(x, \mathtt{tag}) = \mathtt{E.Dec}(\mathtt{sk}_{\mathtt{mac}}, \bar{x})$ and uses the built-in key $\mathtt{ask}$ to verify that $\mathtt{MAC.Veri}(\mathtt{ask}, x, \mathtt{tag})$ is true. If the verification fails, $M_{\mathtt{mac}}$ enters $q_{\mathtt{noop}}$ state and keeps doing nothing until it stops. Otherwise, $M_{\mathtt{mac}}$ computes the value of $M(x)$, chooses a random key $\mathtt{sk}_{\mathtt{o}} = \mathtt{E.KeyGen}(1^\kappa)$ and outputs $M_{\mathtt{mac}}(\bar{x}) = \mathtt{sk}_{\mathtt{o}}||\mathtt{E.Enc}(\mathtt{sk}_{\mathtt{o}}, M(x))$.

The garbling scheme $\mathtt{RGT}_1 = (\mathtt{RGT}_1.\mathtt{Garble}, \mathtt{RGT}_1.\mathtt{Enc}, \mathtt{RGT}_1.\mathtt{Eval})$ for Turing machines M without $\mathtt{ABE}_2$ is then constructed as follows.

$(\mathtt{gsk}, \overline{M}) = \mathtt{RGT}_1.\mathtt{Garble}(1^\kappa, M)$:

- $\mathtt{sk}_{\mathtt{mac}} = \mathtt{E.KeyGen}(1^\kappa)$, $\mathtt{ask} = \mathtt{MAC.KeyGen}(1^\kappa)$, and $(\mathtt{psk}_i, \mathtt{ppk}_i) = \mathtt{PK.KeyGen}(1^\kappa)$ for $i = 0, 1$.

– Construct $M_{\mathtt{mac}}$ from M.
– Define $U_{M_{\mathtt{mac}}}$ and design the reusable garble garbled circuit $\overline{C}_\delta$ for $U_{M_{\mathtt{mac}}}$'s transition function δ as in the process $\mathtt{RGT.Garble}(1^\kappa, U_{M_{\mathtt{mac}}})$ of Section 4. Let $\overline{U}_{M_{\mathtt{mac}}} = (s(t), \overline{C}_\delta)$ be the resulting Turing machine.
– Let $C_d \in NC^1$ be the following circuit:

> **Input:** $U_{M_{\mathtt{mac}}}$'s decrypted output tape $(\mathtt{otape}, \mathtt{session}_b)$ and decrypted control tape $(\mathtt{state}, \mathtt{session}_s)$.
> 1. if $\mathtt{session}_s$ and $\mathtt{session}_b$ are inconsistent then exit.
> 2. let $(\mathtt{sk}_o, \bar{y}) = \mathtt{otape}$.
> 3. compute $y = \mathtt{E.Dec}(\mathtt{sk}_o, \bar{y})$ and output y.

– Let $\overline{C}_d = \mathtt{RGC}_{\mathtt{nc}^1}.\mathtt{Garble}(1^\kappa, \mathtt{ppk}_0, \mathtt{ppk}_1, \mathtt{psk}_0, C_d)$. Here we provide the parameters $\mathtt{ppk}_0, \mathtt{ppk}_1$, and $\mathtt{psk}_0$ to $\mathtt{RGC}_{\mathtt{nc}^1}$ to overwrite the corresponding internal key generation process within $\mathtt{RGC}_{\mathtt{nc}^1}.\mathtt{Garble}$. By overwriting the key generation process for $\mathtt{RGC}_{\mathtt{nc}^1}$, the output of $\overline{U}_{M_{\mathtt{mac}}}$ is in the correct encoding format according to $\mathtt{RGC}_{\mathtt{nc}^1}.\mathtt{Enc}$ and is ready for $\overline{C}_d$ to process.
– Output $\overline{M} = (s(t), \overline{C}_\delta, \overline{C}_d)$ and $\mathtt{gsk} = (\mathtt{ppk}_0, \mathtt{ppk}_1, \mathtt{psk}_0, \mathtt{sk}_{\mathtt{mac}}, \mathtt{ask})$.

$c_x = \mathtt{RGT}_1.\mathtt{Enc}(\mathtt{gsk}, x)$:
– Let $\bar{x} = \mathtt{E.Enc}(\mathtt{sk}_{\mathtt{mac}}, x || \mathtt{MAC.Enc}(\mathtt{ask}, x))$.
– Let c_x be constructed for $\bar{x}$ as in the process $\mathtt{RGT.Enc}(\mathtt{gsk}, \bar{x})$.

$M(x) = \mathtt{RGT}_1.\mathtt{Eval}(\overline{M}, c_x)$:
– Run the Turing machine $\overline{U}_{M_{\mathtt{mac}}} = (s(t), \overline{C}_\delta)$ on input c_x until it stops.
– Run $\overline{C}_d$ on the output tape and control tape of $\overline{U}_{M_{\mathtt{mac}}}$ to obtain $M(x)$.

Comments: We have two comments for the construction of $\mathtt{RGT}_1$.
– The circuit C_d in $\mathtt{RGT}_1$ takes the entire tape as input and decrypts it at the same time. In practice, it may be more efficient to define C_d in a way that it only takes one cell and decrypts the cell separately. In order to achieve this, Turing machine $M_{\mathtt{mac}}$ needs to be revised further so that the output cells are encrypted separately.
– For the convenience of presenting constructions of $\mathtt{RGT}$ and $\mathtt{RGT}_1$ in a compatible way, we constructed C_δ and C_d separately. Indeed, for the construction of $\mathtt{RGT}_1$, C_δ and C_d can be defined as one circuit which is then garbled using Garg et al's NC^1 garbling scheme.

We can then prove the following theorem in a similar way as that of Theorem 2. The proof is omitted in this extended abstract.

Theorem 3. *Assume that the one-time garbling scheme* $\mathtt{GC}$ *is secure, both ciphers* $\mathtt{E.Enc}$ *and* $\mathtt{PK.Enc}$ *are semantically secure, the message authentication scheme* $\mathtt{MAC}$ *is secure, and* $\mathtt{RGC}_{\mathtt{nc}^1}$ *is secure according to Definition 2. Then the reusable garbling scheme* $\mathtt{RGT}_1$ *for Turing machines is secure according to the security definition in Goldwasser et al [17].*

6 Discussions and oblivious Turing machines

In the construction of $\mathtt{RGT}_1$, we used a secure message authentication scheme to protect adversaries from swapping/inserting/deleting/duplicating input tape cells. Some other

techniques could also be used to achieve this same goal. For example, one may use Chaitin's universal self-delimiting Turing machines [11]. A universal self-delimiting Turing machine U takes the input px and outputs $U(px) = M_p(x)$ where p is the encoding of a self-delimiting Turing machine M_p. For a self-delimiting Turing machine M_p, if $M_p(x)$ is defined, then $M_p(y)$ is not defined for all strings y with y being a prefix of x or x being a prefix of y. Before Turing machine stops, it needs to mark each cell on the output tape as final by inserting a special symbol such as FIN to each cell on the output tape. The circuit C_d would only decrypt cells marked as final.

It should also be noted that the black cell "B" could be encoded in advance so that for each input, the Turing machine owner does not need to encode the entire working tape cells. The self-delimiting Turing machines could be used to defeat the attacks that the adversary copies some input cells to some"B" cells and potentially runs the garbled Turing machine on inputs that are not provided by the machine owner.

In our construction of RGT and $\mathtt{RGT}_1$, oblivious Turing machines are used to determine the next cell that the Turing machine needs to process. If Turing machine head movement pattern does not need to be protected, this requirement could be dropped since the circuit C_δ could output the head movement symbol "R" or "L" in plain text.

In Goldwasser et al's garbling scheme [16], the owner of a Turing machine M first converts M to an oblivious Turing machine M_O using the Pippenger-Fischer transformation [23], where an oblivious Turing machine is a Turing machine whose head movement is independent of the current cell content. From M_O, a new Turing machine $M_{\mathtt{FHE}}$ is constructed to perform the FHE evaluation of M_O. The owner of the Turing machine M gives $M_{\mathtt{FHE}}$ to the evaluator. Each time when the Turing machine owner wants the evaluator to calculate $M(x)$, the Turing machine owner creates a homomorphic encryption scheme public key hpk and a corresponding private key hsk. Using the newly created public key hpk, the Turing machine owner calculates the homomorphic encryption cipher texts $c_x = (E_{\mathtt{hpk}}(x[0]), \cdots, E_{\mathtt{hpk}}(x[n-1]))$ for the input $x = x[0] \cdots x[n-1]$ bit by bit and constructs a Yao's one-time garbled circuit D for decrypting the homomorphic encryption scheme by integrating the private key hsk within D. The Turing machine owner then gives $(c_x, D, \mathtt{hpk})$ to the evaluator. The evaluator runs $M_{\mathtt{FHE}}$ on c_x homomorphically step by step. During the evaluation, each cell of M_O's tape corresponds to the FHE ciphertext of $M_{\mathtt{FHE}}$'s cell value and $M_{\mathtt{FHE}}$ maintains the FHE ciphertext $\overline{\mathtt{state}}_i$ of M_O's current state. At step i, $M_{\mathtt{FHE}}$ takes as input the encrypted cell $\bar{b}$ from the input tape that the head currently points at and the current encrypted state $\overline{\mathtt{state}}_i$. Then $M_{\mathtt{FHE}}$ outputs an encrypted new state $\overline{\mathtt{state}}_{i+1}$ and a new content $\bar{b}'$. $M_{\mathtt{FHE}}$ updates the current cell with $\bar{b}'$ and then moves its head left or right according to the oblivious head movement definition. Though [16] did not describe how to get the value $(\overline{\mathtt{state}}_{i+1}, \bar{b}')$ from $(\overline{\mathtt{state}}_i, \bar{b})$. We assume that it uses the straightforward circuit simulation of the Turing machine transition functions. That is, a circuit Π_δ with inputs $(\mathtt{state}_i, b)$ and outputs $(\mathtt{state}_{i+1}, b')$ is constructed from M_O's transition function δ and $M_{\mathtt{FHE}}$ homomorphically evaluates Π_δ to obtain $(\overline{\mathtt{state}}_{i+1}, \bar{b}')$ from $(\overline{\mathtt{state}}_i, \bar{b})$. After the evaluation, the evaluator obtains the homomorphic encryption ciphertext $E_{\mathtt{hpk}}(M(x))$ of $M(x)$. In order for the evaluator to decrypt $E_{\mathtt{hpk}}(M(x))$, the circuit owner uses an attribute based encryption scheme for Turing machines (constructed from the witness encryption scheme) to send corresponding labels for the gar-

bled circuit D so that the evaluator will be able to decrypt $E_{\mathtt{hpk}}(M(x))$ to $M(x)$. In the above scheme, the Turing machine M_O's transition function is leaked via the circuit Π_δ. Though [16] provides no details on how to avoid this leakage, we assume that the authors used the same approach as in Goldwasser et al [17] to protect the privacy of Turing machine M's transition function. That is, $M_{\mathtt{FHE}}$ is constructed for a universal oblivious Turing machine U_O and the description of M is encrypted using an ideal cipher scheme E such as AES. The evaluator only holds the encrypted version $\mathtt{E.Enc}(\mathtt{sk}, M)$ of M. For each evaluation of M on x, the Turing machine owner needs to give both $E_{\mathtt{hpk}}(x)$ and $E_{\mathtt{hpk}}(\mathtt{sk})$ to the evaluator.

7 Conclusion

Using multilinear maps, Garg et al showed the existence of reusable garbling schemes for NC^1 circuits. By further using FHE schemes, Garg et al showed the existence of reusable garbling schemes for all polynomial size circuits. This paper constructed reusable garbling schemes for Turing machines (that is, for all polynomial size circuits also) only assuming the existence of secure multilinear maps. Though it is not clear whether multilinear maps based indistinguishability obfuscators could be more efficient than FHE, the goal of this paper is to develop alternative techniques for resuable garbling schemes to stimulate further research in this direction.

References

1. M. Albrecht and A Davidson. Are graded encoding schemes broken yet? `https://malb.io/are-graded-encoding-schemes-broken-yet.html`.
2. P. Ananth, D. Boneh, S. Garg, A. Sahai, and M. Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.
3. B. Barak, S. Garg, Y.T. Kalai, O. Paneth, and A. Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT 2014*, pages 221–238. Springer, 2014.
4. M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *Proc. 2012 ACM CCS*, pages 784–796. ACM, 2012.
5. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proc. 45th ACM STOC*, pages 111–120. ACM, 2013.
6. N. Bitansky, S. Garg, and S. Telang. Succinct randomized encodings and their applications. Technical report, Cryptology ePrint Archive, Report 2014/771, 2014. http://eprint. iacr. org, 2014.
7. E. Boyle, K. Chung, and R. Pass. On extractability obfuscation. In *Theory of Cryptography*, pages 52–73. Springer, 2014.
8. Z. Brakerski and G. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *Theory of Cryptography*, pages 1–25. Springer, 2014.
9. R. Canetti, J. Holmgren, A. Jain, and V. Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and ram programs. In *Proc. STOC 15*, New York, NY, USA, 2015. ACM.
10. Ran Canetti and Justin Holmgren. Fully succinct garbled ram. 2015.
11. G. J. Chaitin. On the length of programs for computing finite binary sequences. *J. Assoc. Comput. Math.*, 13:547–569, 1966.
12. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proc. IEEE 54th FOCS*, pages 40–49. IEEE, 2013.

13. S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In *Proc. 45th ACM STOC*, pages 467–476. ACM, 2013.
14. S. Garg, S. Lu, R. Ostrovsky, and A. Scafuro. Garbled ram from one-way functions. In *Proc. STOC 15*, New York, NY, USA, 2015. ACM.
15. C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs. Garbled ram revisited. In *EUROCRYPT*, pages 405–422. Springer, 2014.
16. S. Goldwasser, Y. Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run Turing machines on encrypted data. In *Proc. CRYPTO*, pages 536–553. 2013.
17. S. Goldwasser, Y. Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proc. 45th STOC*, pages 555–564. ACM, 2013.
18. Ming-Deh A Huang. Trilinear maps for cryptography. *arXiv preprint arXiv:1803.10325*, 2018.
19. Ming-Deh A Huang. Trilinear maps for cryptography ii. *arXiv preprint arXiv:1810.03646*, 2018.
20. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *Proc. 40th ACM STOC*, pages 433–442. ACM, 2008.
21. V. Koppula, A. Lewko, and B. Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In *Proc. STOC 15*, New York, NY, USA, 2015. ACM.
22. H. Lin and R. Pass. Succinct garbling schemes and applications. Technical report, Cryptology ePrint Archive, Report 2014/766, 2014. http://eprint. iacr. org, 2014.
23. N. Pippenger and M. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.
24. A. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE FOCS*, pages 162–167. IEEE, 1986.