Indistinguishability Obfuscation with Constant Size Overhead

Prabhanjan Ananth* Abhishek Jain† Amit Sahai‡

Abstract

Present constructions of indistinguishability obfuscation (i$\mathcal{O}$) create obfuscated programs where the size of the obfuscated program is at least a factor of a security parameter larger than the size of the original program.

In this work, we construct the first i$\mathcal{O}$ scheme that achieves only a *constant* multiplicative overhead (in fact, the constant is 2) in the size of the program. The security of our construction requires the existence of sub-exponentially secure i$\mathcal{O}$ for circuits (that has any polynomial multiplicative overhead in the circuit size) and one-way functions.

1 Introduction

The emergence of indistinguishability obfuscation [7] (i$\mathcal{O}$) has revolutionized cryptography. Since the work of Garg et al. [22] who presented its first candidate construction, i$\mathcal{O}$ has been used to realize numerous advanced cryptographic tasks, such as functional encryption [22] and deniable encryption [40], that previously seemed beyond our reach. Indeed, by now, i$\mathcal{O}$ has been firmly established as a central hub for cryptography.

The existing mechanisms for general-purpose i$\mathcal{O}$, however, are highly inefficient in terms of the *size* of the obfuscated programs that are produced. In particular, all known mechanisms for i$\mathcal{O}$ yield obfuscated programs of size polynomial in the *size of the underlying program and the security parameter*. In other words, they incur a multiplicative overhead of at least the security parameter. More concretely, prior works on constructing i$\mathcal{O}$ can be divided into the following two categories:

- *Obfuscating Circuits*: By now, there is a large sequence of works that provide candidate constructions of i$\mathcal{O}$ for general *circuits* [22, 6, 16, 37, 25, 2, 44, 41, 5]. However, in all of these works, the obfuscation of a circuit C is of the size poly$(\lambda, |C|)$, where the size of the obfuscation grows at least linearly with the security parameter times the size of the circuit.
- *Obfuscating Turing machines*: Another body of work has focused on the problem of obfuscating Turing machines directly [14, 1, 31, 9, 18, 33, 17, 20].[1] Moving to the Turing

*University of California Los Angeles. Email: prabhanjan@cs.ucla.edu. This work was partially supported by grant #360584 from the Simons Foundation.

†Johns Hopkins University. Email: abhishek@cs.jhu.edu. Supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213 and NSF CNS-1414023.

‡University of California Los Angeles. Email: sahai@cs.ucla.edu Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

[1]We remark that the works of [9, 18, 17, 20] also study the problem of directly obfuscating RAM programs. In this work, we restrict our attention to the Turing Machine model.

Machine model yields significant efficiency improvements over the circuit model since the size of the Turing Machine may be much smaller than the corresponding circuit size. Furthermore, working in the Turing Machine model yields the benefit of achieving per-input running time, as opposed to incurring worst-case running time that is inherent to the circuit model of computation.

Nevertheless, we note that in all of these works, the size of the obfuscation of a Turing Machine M is at least $\mathrm{poly}(\lambda, |M|)$, where the size of the obfuscation grows at least linearly with the security parameter times the size of the Turing Machine description. In particular, [14, 1, 31] achieve these parameters by relying on (public-coin) differing inputs obfuscation [7, 31]. In contrast, the works of [9, 18, 33] only make use of $\mathsf{i}\mathcal{O}$ for circuits; however, these works are restricted to Turing machines with bounded-length inputs, and as such incur overhead of $\mathrm{poly}(\lambda, |M|, L)$, where L is the bound on the input length.[2]

Thus, in summary, irrespective of the model of the computation, a multiplicative dependence on the underlying program size and the security parameter is inherent to the obfuscated program size in all the prior works.

Our Goal. In this work, we ask the question:

Is it possible to realize general-purpose $\mathsf{i}\mathcal{O}$ with
constant multiplicative overhead in program size?

While this question is already meaningful in the circuit model of computation, we will focus on the Turing Machine model. In particular, we ask the question whether it is possible to obfuscate bounded-input Turing Machines such that the resulting machine is of size $c \cdot |M| + \mathrm{poly}(\lambda, L)$, where c is a universal constant and L is the input length bound.

Achieving constant multiplicative overhead has been a major goal in many areas of computer science, from constructing asymptotically good error correcting codes, to encryption schemes where the size of the ciphertext is linear in the size of the plaintext. To the best of our knowledge, however, this question in the context of program obfuscation has appeared to be far out of reach in the context of basing security on $\mathsf{i}\mathcal{O}$ itself.[3]

Bounded-input vs Unbounded-input Turing machines. We note that if we could build $\mathsf{i}\mathcal{O}$ for Turing machines with *unbounded* input length, then the question of constant overhead in size is moot: indeed, one could simply obfuscate a universal Turing machine and pass on the actual machine that one wishes to obfuscate as an (encrypted) input. The state of the art in $\mathsf{i}\mathcal{O}$ research, however, is still limited to Turing machines with *bounded* input length. In this case, the above approach does not work since the size of the obfuscation for bounded-input TMs grows polynomially in the input length bound which would yield a polynomial overhead in the size of the Turing machine that we wish to obfuscate.

In a subsequent work, [34] provide a transformation from output compressing randomized encodings for TMs to $\mathsf{i}\mathcal{O}$ for unbounded-input TMs. However, no construction (with a security reduction) is presently known for such randomized encodings. In particular, in the same work, [34] show that such randomized encodings, in general, do not exist.

[2]We note that [9, 18], in fact, only work for memory-bounded Turing machines, and hence incur additional overhead in the maximum memory size of the Turing machine. The work of [33] does not suffer from this restriction.

[3]We observe that using (public-coin) differing input obfuscation, a variant of the construction given by [14, 1, 31] where FHE is combined with hybrid encryption, can yield constant multiplicative overhead. However, the plausibility of differing input obfuscation has come under scrutiny [23], and unlike for indistinguishability obfuscation [37, 25], there are no known security reductions supporting the existence of differing input obfuscation. Nor are there constructions of differing input obfuscation from other natural primitives, analogous to recent constructions of indistinguishability obfuscation from compact functional encryption [3, 10]. Thus, in this work, we focus only on achieving $\mathsf{i}\mathcal{O}$ with constant multiplicative overhead from the existence of $\mathsf{i}\mathcal{O}$ (without constant multiplicative overhead) itself.

1.1 Our Results

$\mathsf{i}\mathcal{O}$ with Constant Multiplicative Overhead. In this work, we resolve the aforementioned question in the affirmative. We provide a construction of $\mathsf{i}\mathcal{O}$ for Turing machines with bounded input length where the size of obfuscation of a machine M is only $2|M|+\text{poly}(\lambda, L)$, where L is the input length bound. Our construction is based on sub-exponentially secure $\mathsf{i}\mathcal{O}$ (with polynomial blowup) for general circuits and one-way functions.

Theorem 1 (Informal)**.** *Assuming sub-exponentially secure $\mathsf{i}\mathcal{O}$ for general circuits and sub-exponentially secure one way functions, there exists an $\mathsf{i}\mathcal{O}$ for Turing Machines with bounded input length such that the size of the obfuscation of a Turing machine M is $2\cdot|M|+\text{poly}(\lambda, L)$, where L is an input length bound.*

KLW Simplification. Our approach to establish Theorem 1, in fact, yields a conceptually simpler variant of the recent work of Koppula, Lewko and Waters (KLW)[33] who gave the first construction of $\mathsf{i}\mathcal{O}$ for Turing machines with unbounded memory (but bounded-length inputs).

In order to obtain their main result, [33] use an intermediate notion of *machine-hiding* encodings which in turns uses several novel ideas, in particular, a special hash function (referred to as positional accumulator) and a "reverse hybrid" proof strategy. They also consider the weaker notion of *message-hiding* encodings which still requires the special hash function but not the reverse hybrid strategy. However, this weaker primitive does not suffice for their construction of $\mathsf{i}\mathcal{O}$ for TMs.

At a high-level, the crucial difference between message-hiding encodings and machine-hiding encodings is that the latter concerns with hiding the computation of a TM while the former only deals with authentication property. Nevertheless, we show how to achieve $\mathsf{i}\mathcal{O}$ for TMs directly from message-hiding encodings. This yields a conceptually simpler variant of their result with a simpler proof.

Applications. Our result and the techniques used therein can be applied in many application scenarios to achieve commensurate efficiency gains. Below we highlight some of these applications.

I. Functional Encryption with Constant Multiplicative Overhead. Plugging in our $\mathsf{i}\mathcal{O}$ in the functional encryption (FE) scheme of Waters [42],[4] we obtain the first FE scheme for Turing machines where the size of a function key for a turing machine M with input length bound L is only $c \cdot |M| + \text{poly}(\lambda, L)$ for some constant c. Further, the size of a ciphertext for any message x is only $c' \cdot |x| + \text{poly}(\lambda)$ for some constant c'.[5]

II. Unbounded input FE. The size of the function keys can be further reduced by leveraging the recent result of [4] who construct adaptively secure FE for TMs with *unbounded* length inputs, based on $\mathsf{i}\mathcal{O}$ and one-way functions. Instantiating their FE construction with our $\mathsf{i}\mathcal{O}$ and the above discussed FE scheme, we obtain the first construction of an (adaptively secure) FE scheme where the size of a function key for an unbounded length input TM M is only $c \cdot |M| + \text{poly}(\lambda)$ for a small constant c.

III. Reusable Garbled Turing Machines with Constant Overhead. By applying the transformation of De Caro et al. [19] on the above FE scheme, we obtain a simulation-secure FE scheme with constant multiplicative overhead. Next, by applying the transformation of Goldwasser et al. [27] on the simulation-secure FE scheme, we obtain the first construction of reusable garbled Turing Machine scheme where both the machine encodings and input encodings incur only constant multiplicative overhead in the size of the machine and input, respectively. Specifically, the encoding size of a machine M is $c \cdot |M| + \text{poly}(\lambda)$, while the encoding size of an input x is $c_1 \cdot |x| + c_2|M(x)| + \text{poly}(\lambda)$ for some constants c, c_1, c_2.

[4][42] presents two FE schemes: the first one only handles post-challenge key queries, while the second one allows for both pre-challenge and post-challenge key queries. We only consider the instantiation of the first scheme with our $\mathsf{i}\mathcal{O}$.

[5]The construction of [42] already achieves the second property.

Previously, Boneh et al. [11] constructed reusable garbled *circuits* with additive overhead in either the circuit encoding size, or the input encoding size (but not both simultaneously).

IV. Publicly Verifiable Delegation of Computation with Low Communication. Plugging in our (reusable) garbled turing machine in the delegation protocol of [18], we obtain a two-round publicly verifiable delegation of computation protocol where the communication complexity only incurs a constant multiplicative overhead in the size of the TM whose computation is being delegated. Very briefly, to delegate the computation of a machine M on an input x, the delegator samples a key pair (sk, vk) of a signature scheme and sends a garbling of (M', x) to the worker, where machine M' computes $y = M(x)$ and outputs $(y, Sign(sk, y))$. The result is accepted by the delegator if the signature verifies w.r.t. vk.

From the efficiency of the (reusable) garbled turing machine, it follows that the communication complexity of the above protocol is only $c_1 \cdot |M| + c_2 \cdot |x| + c_3 \cdot |M(x)| + \text{poly}(\lambda)$ for some small constants c_1, c_2, c_3. This protocol, in fact, only requires a one-time garbled Turing machine, and by using a *reusable* garbled Turing machine and a one-time pre-processing phase, the communication complexity can be amortized.

1.2 Technical Overview

We start by recalling the common template for constructing i$\mathcal{O}$ for Turing machines (TM) used in the recent works of [9, 18, 33]. We note that all of these works are restricted to TMs with inputs of a priori bounded length, and we will also consider this restricted setting. For simplicity of discussion, however, we will ignore this restriction in this section.

Prior work: a two-step approach. [9, 18, 33] reduce the problem of obfuscating Turing machines to the problem of obfuscating circuits. This is achieved in the following two steps:

1. *Randomized encoding for TMs.* First, using i$\mathcal{O}$ for circuits, they construct a randomized encoding (RE) for Turing machines.[6]
2. *From RE to i$\mathcal{O}$.* In the second step, RE for TMs is combined with i$\mathcal{O}$ for circuits to obtain i$\mathcal{O}$ for TMs.[7] Very roughly, the obfuscation of a machine M corresponds to obfuscation of a circuit C_M (that has M hardwired). On input x, C_M outputs a RE of $M(x)$. To recover $M(x)$, the evaluator simply executes the decoding algorithm of RE.

The above approach, however, is highly problematic in our setting. Recall that our goal is to construct i$\mathcal{O}$ for TMs with constant multiplicative overhead in the size of the TM. Then, zooming in on the second step, note that even if we start with an RE for TMs with constant multiplicative overhead in size, in order to achieve our goal, we will need the i$\mathcal{O}$ for circuits used in this step to already satisfy the constant multiplicative overhead property! An even more serious issue is that we will also require the *running time* of the RE encode procedure to have only a constant multiplicative overhead in its input size, namely, $|M| + |x|$. Indeed, ensuring that the running time has only a constant multiplicative overhead in the input size is in general a hard problem for many cryptographic primitives (see [30] for discussion).

Towards that end, we devise a new approach to achieve our goal. We describe it in the remainder of this section.

Oblivious Evaluation Encodings (OEE). The protagonist of our construction is a new primitive that we refer to as *oblivious evaluation encodings* (OEE). We explain the notion with a simple illustration: Alice wishes to delegate her computation to Bob. However, she is undecided whether to delegate the computation of machine M_0 on x or delegate computation of M_1 on x. One option for Alice would be that she delegates both the computations. However in this scenario, Bob learns both $M_0(x)$ and $M_1(x)$. Oblivious evaluation encodings offers a solution to this problem. It allows Alice to encode (M_0, M_1) together and send it across to Bob. Later when Alice has made up her mind, she can send the encoding of (x, b),

[6]In [9, 18], the size of the randomized encoding of $M(x)$ for a machine M and input x depends on the amount of space required in the computation of $M(x)$. The construction of [33] does not have this limitation.

[7]As noted in [33], it suffices to use the weaker notion of *machine hiding encoding* in this transformation.

where M_b is the machine Alice wants Bob to execute. Bob with the help of the encodings (x, b) and (M_0, M_1) can recover $M_b(x)$.

The key algorithms in a OEE scheme are described below:

- Setup: Generate a secret key sk.
- Input encoding: On input x and a "choice" bit b, generate an encoding of (x, b).
- Turing machine encoding: On input two Turing machines M_0 and M_1, generate an encoding of (M_0, M_1).
- Decode: On input encodings of (x, b) and (M_0, M_1), output $M_b(x)$.

An OEE scheme with constant multiplicative overhead is one where the size of the encoding is $|M_0| + |M_1| + \text{poly}(\lambda)$.

An informed reader might find some similarities between OEE and oblivious transfer [39, 21]. Indeed, the name for our primitive is inspired by oblivious transfer.

It turns out that for our main result, the above notion itself will not suffice. We augment it with helper *key puncturing* algorithms as described below. The exact roles of these algorithms will become clear later when we describe our construction of $\mathsf{i}\mathcal{O}$ for TMs with constant multiplicative overhead using OEE.

- Input puncturing: On input secret key sk (produced by Setup) and input x, it outputs a secret key sk_x^{inp}. This punctured secret key allows for the computation of an encoding of $(x', 0)$ and $(x', 1)$ for all inputs $x' \neq x$.
- (Choice) bit puncturing: On input secret key sk and bit b, it outputs a secret key sk_b^{bit}. This punctured secret key allows to compute an encoding of (x, b) for all x. However, an adversary should not be able to compute encoding of an input x with choice bit $\bar{b}$.

Our main technical contribution is two fold: first, we give a construction of $\mathsf{i}\mathcal{O}$ for TMs with constant multiplicative overhead from $\mathsf{i}\mathcal{O}$ for circuits (with polynomial overhead) and OEE with constant multiplicative overhead. Our second contribution is the construction of such an OEE.

$\mathsf{i}\mathcal{O}$ for TMs from OEE. We give a high level description of the construction. An obfuscation of a machine M consists of two values: (a) An OEE TM encoding of (M, M) generated using an OEE secret key sk. (b) Indistinguishability obfuscation of a circuit $C_{sk,K}$ that on input x outputs an OEE input encoding of $(x, 0)$. The circuit $C_{sk,K}$ has hardwired in it the OEE key sk and a PRF key K to generate the randomness required in the input encoding procedure. Evaluation on input x proceeds by first computing the encoding of $(x, 0)$ and then decoding the encodings $(x, 0)$ and (M, M), using the OEE decode algorithm, to obtain $M(x)$.

Note that unlike the approach of [9, 18, 33] where a fresh RE is computed on-the-fly each time the user wishes to evaluate the obfuscated program on an input, here, the same TM encoding is *reused* for each evaluation. In particular, only the input encoding is computed on-the-fly. Because of this crucial difference, if we instantiate the above construction with an OEE scheme with constant multiplicative overhead, then the resulting obfuscation scheme also satisfies the same property *even* if the obfuscation of circuit $C_{sk,K}$ has polynomial overhead in size.

The main remaining challenge is to argue security. Consider two machines M_0 and M_1 that are functionally equivalent. We need to argue that obfuscations of M_0 and M_1 are computationally indistinguishable. We sketch the hybrids in the security proof below, highlighting the use of the key puncturing properties of OEE.

We start off with the hybrid where an honestly generated obfuscation of M_0 is given to the adversary. Using a sequence of intermediate hybrids described below, we reach the final hybrid which corresponds to the obfuscation of M_1.

1. The secret key sk hardwired in $C_{sk,K}$ is replaced by the punctured key sk_0^{bit}. The functionality of the circuit does not change and hence we can invoke the security of iO for circuits here.

2. We change the OEE TM encoding of (M_0, M_0) to an encoding of (M_0, M_1). The indistinguishability of this step follows from the security property associated with the (choice) bit encoding property.

3. We then define an intermediate sequence of 2^n hybrids[8]: in the i^{th} hybrid, the obfuscated circuit computes input encoding of $(x, 1)$ for all $x < i$ and $(x, 0)$ for all $x \geq i$. To switch from i^{th} hybrid to $(i+1)^{th}$ hybrid, we crucially use the input puncturing property of OEE.

4. Upon the completion of the 2^n intermediate hybrids, we have the obfuscated circuit producing only input encodings of $(x, 1)$ for all inputs x. We then replace the key in the obfuscated circuit with the punctured key sk_1^{bit}.

5. We can now safely switch the TM encoding of (M_0, M_1) to (M_1, M_1), again using the security property of (choice) bit encoding. This corresponds to the obfuscation of M_1.

The only remaining piece in the puzzle is the construction of OEE with constant multiplicative overhead.

Construction of OEE from ABE for TMs. The main tool in our construction is a public key attribute based encryption (ABE) for TMs with constant multiplicative overhead[9]. ABE for Turing machines is defined in the same manner as ABE for circuits (see e.g., [29]), except that the attribute keys are now associated to TMs instead of circuits. The constant multiplicative overhead property requires that the size of an ABE key of a TM M is $c \cdot |M| + \mathrm{poly}(\lambda)$, where c is a constant.

We start by giving a construction of OEE from ABE for TMs. Later we will sketch our construction of ABE from TMs.

At a first glance, it is not apparent how the primitives OEE and ABE are related to each other. ABE allows for conditional disclosure of messages from ciphertexts whereas OEE serves the purpose of "delayed" delegation (the choice of which machine needs to be delegated is delayed). The most basic difference is that in ABE, computation is not hidden whereas in the case of OEE, the machine and hence the computation needs to be hidden. But ABE offers a way of authenticating computation that is implicitly required in an OEE scheme. Indeed we utilize this aspect of ABE to obtain a construction for OEE.

As a starting point, we encode the pair of machines (M_0, M_1) by first encrypting them together. Since we perform computation on the machines, the encryption scheme we use is fully homomorphic [24]. In the input encoding of (x, b), we encrypt the choice bit b using the same public key. To evaluate (M_0, M_1) on (x, b), we execute the homomorphic evaluation function. Notice, however, that the output is in encrypted form. We need to provide additional capability to the evaluator to decrypt the output (and nothing else). One way around is that the input encoding algorithm publishes a garbling of the FHE decryption algorithm. But the input encoder must somehow convey the garbled circuit wire keys, corresponding to the output of the FHE evaluation, to the evaluator.

This is where ABE for TMs comes to the rescue. Using ABE, we can ensure that the evaluator gets *only* the wire keys corresponding to the output of the FHE evaluation. Once this is achieved, the garbled circuit that is provided as part of the input encoding can then be evaluated to obtain the decrypted output. We can then show that the resulting OEE scheme has constant multiplicative overhead if the underlying ABE scheme also satisfies this property.

This approach of using ABE, FHE and garbled circuits in the above manner is inspired by the work of Goldwasser et al. [27]. We note, however, that their work used these techniques in a different context (that of obtaining a succinct single-key FE). Furthermore, there are differences in the way we use these tools. In particular, in our case FHE parameters are crucially *reused* across different encodings, while in their case, a fresh instantiation of FHE is necessarily used for every execution.

[8]Here 2^n is the total number of inputs to the machines M_0 and M_1.

[9]We in fact construct an ABE scheme for TMs with *additive* overhead. For our main result, however, ABE for TMs with constant multiplicative overhead suffices.

While the above high level idea is promising, there are still some serious issues. The first issue is that we need to homomorphically evaluate on Turing machines as against circuits. This can be resolved by using the powers-of-two evaluation technique from the work of [28]. The second and the more important question is: what are the punctured keys? The input puncturing key could simply be the ABE public key and the FHE public key-secret key pair. The choice bit puncturing key, however, is more tricky. Note that setting the FHE secret key to be the punctured key will ensure correctness but completely destroy the security. To resolve this issue, we use the classic two-key technique [36]. We encrypt machines M_0 and M_1 using two different FHE public keys. The choice bit puncturing key would just be one of the FHE secret keys depending on which bit needs to be punctured. For more details, we refer the reader to the technical sections.

ABE for TMs with Constant Multiplicative Overhead. We now shift gears and focus on constructing ABE for TMs with constant multiplicative overhead. Our starting point is the message hiding encoding scheme of Koppula et al. [33]. A message hiding scheme allows for encoding a secret that is associated to a TM-input pair (M, x) such that the secret is revealed only if $M(x) = 1$. While this notion is similar in spirit to ABE, the notion of ABE is more general than message hiding schemes: (i) ABE has a *decomposability* property, i.e., it allows for encoding M (ABE key of M) and (x, secret) separately while they are encoded together in a message hiding scheme and (ii) ABE offers *reusability*: the encoding of M can be reused for different encodings (encryptions) of (x, secret). Despite these fundamental differences, we show that the main tools of KLW can still be used to achieve our goal.

We describe the basic idea at a high level and leave the details to later sections. The basic framework of KLW is as follows: the message hiding encoding comprises of a public storage tree computed on the input x along with an obfuscated program that computes the next message function of M. The root of the storage tree is authenticated using a special type of signature scheme, called *splittable signatures*. The obfuscated program has hardwired into it the secret that is revealed only when the accepted state is reached.

To obtain an ABE scheme, we first observe that the encoding of the KLW scheme already has some form of decomposability – the storage tree along with the authentication on the root can be thought of as being the input encoding and the obfuscated program can be thought of as being the TM encoding. However we require that the input and the secret (part of the obfuscated program) are encoded together but here, the machine and the secret are coupled together. To solve this, we *reverse* the roles of the input and the TMs – we now compute the storage tree on the TM and the obfuscated program contains a universal TM with the input hardwired into it. This reversal will also buy us efficiency as will be evident later. The problem of reusability is relatively harder to deal with: the main reason boils down to the fact that the parameters of the splittable signatures scheme, as guaranteed by the security of KLW, can be used only for one computation.

Signature Synchornization. We provide a *signature synchronization mechanism* that solves the reusability problem. In this mechanism, there is a master signature instantiation used in producing splittable signatures which forms a part of the ABE key. In every execution of the encryption algorithm, a fresh instantiation of the splittable signatures is produced – this ensures that any particular instantiation is not reused. The main issue now is to ensure that the signatures of the master signature instantiation in the ABE key are synchronized with the signatures that are part of the ciphertexts. To do this, we introduce a *translation* box that transforms signatures w.r.t master instantiation into signatures in the ciphertexts. We defer the actual implementation of this mechanism to the technical section.

Finally, we need to ensure that the ABE scheme satisfies the constant multiplicative overhead property. Recall that the ABE key of a machine M comprises of a public storage tree on M along with an authentication on the root of the tree. We observe that the storage tree need not be part of the ABE key at all. Indeed, we will simply set our ABE key to be the root of the storage tree and a signature on it! The evaluator can then reconstruct the storage tree by itself and then utilize the key as before.

2 Preliminaries

We assume familiarity of the reader with the standard cryptography notions. In the Appendix, we recall the notions of Turing machines (Section A.1), puncturable pseudorandom functions (Section A.2), garbling schemes (Section A.3) and fully homomorphic encryption schemes (Section A.4).

Indistinguishability Obfuscation. The notion of indistinguishability obfuscation (iO), first conceived by Barak et al. [7], guarantees that the obfuscation of two circuits are computationally indistinguishable as long as they both are equivalent circuits, i.e., the output of both the circuits are the same on every input. Formally,

Definition 1 (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform PPT algorithm* $\mathsf{i}\mathcal{O}$ *is called an indistinguishability obfuscator for a circuit family* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{C}_\lambda$ *consists of circuits* C *of the form* $C : \{0,1\}^{\mathsf{inp}} \to \{0,1\}$ *with* $\mathsf{inp} = \mathsf{inp}(\lambda)$*, if the following holds:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, every* $C \in \mathcal{C}_\lambda$*, every input* $x \in \{0,1\}^{\mathsf{inp}}$*, we have that*

$$\mathsf{Pr}\left[C'(x) = C(x) \;:\; C' \leftarrow \mathsf{i}\mathcal{O}(\lambda, C)\right] = 1$$

- **Indistinguishability:** *For any PPT distinguisher* D*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: for all sufficiently large* $\lambda \in \mathbb{N}$*, for all pairs of circuits* $C_0, C_1 \in \mathcal{C}_\lambda$ *such that* $C_0(x) = C_1(x)$ *for all inputs* $x \in \{0,1\}^{\mathsf{inp}}$*, we have:*

$$\Big|\mathsf{Pr}\left[D(\lambda, \mathsf{i}\mathcal{O}(\lambda, C_0)) = 1\right] - \mathsf{Pr}[D(\lambda, \mathsf{i}\mathcal{O}(\lambda, C_1)) = 1]\Big| \leq \mathsf{negl}(\lambda)$$

We can additionally enforce the size of the obfuscation of a circuit $C \in \mathcal{C}_\lambda$ to be $c \cdot |C| + \text{poly}(\mathsf{inp}, \lambda)$, where c is a constant. If an obfuscation scheme satisfies this property then we term such an obfuscation scheme as *iO with constant multiplicative overhead.*

This is formally defined below.

Definition 2 (iO for Circuits with Constant Multiplicative Overhead)**.** *An indistinguishability obfuscation scheme,* $\mathsf{i}\mathcal{O}$*, defined for a circuit family* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *is said to be* **iO with constant multiplicative overhead** *if there exists a universal constant* c*, such that for every security parameter* $\lambda \in \mathbb{N}$*, for every* $C \in \mathcal{C}_\lambda$ *with* inp *being the input length of* C*,*

$$|\mathsf{i}\mathcal{O}(\lambda, C)| = c \cdot |C| + \text{poly}(\mathsf{inp}, \lambda)$$

iO for Turing Machines. Analogous to the case of circuits, we can define indistinguishability obfuscation for Turing machines (TMs). We work in a weaker setting of iO for TMs, as considered by the recent works [18, 9, 33], where the inputs to the TM are upper bounded by a pre-determined value. This definition of iO for TMs is referred as *succinct iO*. The security property of this notion states that the obfuscations of two machines M_0 and M_1 are computationally indistinguishable as long as $M_0(x) = M_1(x)$ and the time taken by both the machines on input x are the same, i.e., $\mathsf{RunTime}(M_0, x) = \mathsf{RunTime}(M_1, x)$. The succinctness property ensures that both the obfuscation and the evaluation algorithms are independent of the worst case running times.

As in the case of circuits, here too we can enforce the size of obfuscation of a Turing machine M to be $k \cdot |M| + \text{poly}(\lambda, L)$, where k is a constant and L is the upper bound on the input lengths . A succinct obfuscation satisfying this property is termed as *succinct iO with constant multiplicative overhead.* We formally define this below.

Definition 3 (Succinct iO with Constant Multiplicative Overhead)**.** *A uniform PPT algorithm* SuccIO *is called an succinct indistinguishability obfuscator for a class of Turing machines* $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ *with an input bound* L*, if the following holds:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, every* $M \in \mathcal{M}_\lambda$*, every input* $x \in \{0,1\}^{\leq L}$*, we have that*

$$\mathsf{Pr}\left[M'(x) = M(x) \;:\; M' \leftarrow \mathsf{SuccIO}(\lambda, M, L)\right] = 1$$

- **Indistinguishability:** *For any PPT distinguisher D, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: for all sufficiently large $\lambda \in \mathbb{N}$, for all pairs of Turing machines $M_0, M_1 \in \mathcal{M}_\lambda$ such that $M_0(x) = M_1(x)$ for all inputs $x \in \{0,1\}^{\leq L}$, we have:*
$$\Big|\mathsf{Pr}\left[D(\lambda, \mathsf{SuccIO}(\lambda, M_0, L)) = 1\right] - \mathsf{Pr}[D(\lambda, \mathsf{SuccIO}(\lambda, M_1, L)) = 1]\Big| \leq \mathsf{negl}(\lambda)$$
- **Succinctness:** *For every $\lambda \in \mathbb{N}$, every $M \in \mathcal{M}_\lambda$, we have the running time of* SuccIO *on input (λ, M, L) to be* $\mathrm{poly}(\lambda, |M|, L)$ *and the evaluation time of $\widetilde{M}$ on input x, where $|x| \leq L$, to be* $\mathrm{poly}(|M|, L, t)$*, where $\widetilde{M} \leftarrow \mathsf{SuccIO}(\lambda, M, L)$ and $t = \mathsf{RunTime}(M, x)$.*
- **Constant Multiplicative Overhead:** *There exists a universal constant c such that for every $\lambda \in \mathbb{N}$, for every $M \in \mathcal{M}_\lambda$, we have $|\mathsf{SuccIO}(\lambda, M, L)| = c \cdot |M| + \mathrm{poly}(\lambda, L)$.*

2.1 Building Blocks of KLW [33]

We recall some notions introduced in the work of Koppula, Lewko, Waters [33]. There are three main building blocks: positional accumulators, splittable signatures and iterators. We explain the high level intuition of these three primitives later in the technical sections. The following definitions are stated verbatim from [33]. We do not state the security properties of these three primitives since they are never explicitly used in our work. We reduce the security of our construction in a black box manner to the security of the KLW scheme.

I. Positional Accumulators. A positional accumulator for message space Msg_λ consists of the following algorithms.

$\mathsf{SetupAcc}(1^\lambda, T) \rightarrow (\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0)$: The setup algorithm takes as input a security parameter λ in unary and an integer T in binary representing the maximum number of values that can stored. It outputs public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an initial accumulator value w_0, and an initial storage value $store_0$.

$\mathsf{EnforceRead}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k), \mathsf{ind}^*) \rightarrow (\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0)$: The setup enforce read algorithm takes as input a security parameter λ in unary, an integer T in binary representing the maximum number of values that can be stored, and a sequence of symbol, index pairs, where each index is between 0 and $T-1$, and an additional ind^* also between 0 and $T-1$. It outputs public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an initial accumulator value w_0, and an initial storage value $store_0$.

$\mathsf{EnforceWrite}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k)) \rightarrow (\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0)$: The setup enforce write algorithm takes as input a security parameter λ in unary, an integer T in binary representing the maximum number of values that can be stored, and a sequence of symbol, index pairs, where each index is between 0 and $T-1$. It outputs public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an initial accumulator value w_0, and an initial storage value $store_0$.

$\mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathsf{ind}) \rightarrow (m, \pi)$: The prep-read algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, a storage value $store_{in}$, and an index between 0 and $T-1$. It outputs a symbol m (that can be ϵ) and a value π.

$\mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathsf{ind}) \rightarrow aux$: The prep-write algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, a storage value $store_{in}$, and an index between 0 and $T-1$. It outputs an auxiliary value aux.

$\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{read}, \mathsf{ind}, \pi) \rightarrow (\{True, False\})$: The verify-read algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an accumulator value w_{in}, a symbol, m_{read}, an index between 0 and $T-1$, and a value π. It outputs $True$ or $False$.

$\mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathsf{ind}, m) \rightarrow store_{out}$: The write-store algorithm takes in the public parameters, a storage value $store_{in}$, an index between 0 and $T-1$, and a symbol m. It outputs a storage value $store_{out}$.

$\mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{write}, \mathsf{ind}, aux) \rightarrow (w_{out}$ **or** $Reject)$: The update algorithm takes in the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an accumulator value w_{in}, a symbol m_{write}, and index between 0 and $T-1$, and an auxiliary value aux. It outputs an accumulator value w_{out} or $Reject$.

Remark 1. *In our construction, we will set $T = 2^\lambda$ and so T will not be an explicit input to all the algorithms.*

Correctness. We consider any sequence $(m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k)$ of symbols $m_1, \ldots, m_k$ and indices $\mathsf{ind}_1, \ldots, \mathsf{ind}_k$ each between 0 and $T-1$. We fix any $\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0 \leftarrow \mathsf{SetupAcc}(1^\lambda, T)$. For j from 1 to k, we define $store_j$ iteratively as $store_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{ind}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{j-1}, \mathsf{ind}_j)$ and $w_j := Update(\mathsf{PP}_{\mathsf{Acc}}, w_{j-1}, m_j, \mathsf{ind}_j, aux_j)$. Note that the algorithms other than $\mathsf{SetupAcc}$ are deterministic, so these definitions fix precise values, not random values (conditioned on the fixed starting values $\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$).

Efficiency. The accumulator and π values should have size polynomial in the security parameter λ and $\log(T)$, so Verify-Read and Update will also run in time polynomial in λ and $\log(T)$. Storage values will have size polynomial in the number of values stored so far.

II. Splittable Signatures. The syntax of the splittable signatures scheme is described below.

Syntax. A splittable signature scheme $\mathsf{SplScheme}$ for message space Msg consists of the following algorithms:

$\mathsf{SetupSpl}(1^\lambda)$ The setup algorithm is a randomized algorithm that takes as input the security parameter λ and outputs a signing key SK, a verification key VK and *reject-verification key* $\mathsf{VK}_{\mathsf{rej}}$.

$\mathsf{SignSpl}(\mathsf{SK}, m)$ The signing algorithm is a deterministic algorithm that takes as input a signing key SK and a message $m \in \mathsf{Msg}$. It outputs a signature σ.

$\mathsf{VerSpl}(\mathsf{VK}, m, \sigma)$ The verification algorithm is a deterministic algorithm that takes as input a verification key VK, signature σ and a message m. It outputs either 0 or 1.

$\mathsf{SplitSpl}(\mathsf{SK}, m^*)$ The splitting algorithm is randomized. It takes as input a secret key SK and a message $m^* \in \mathsf{Msg}$. It outputs a signature $\sigma_{\mathsf{one}} = \mathsf{SignSpl}(\mathsf{SK}, m^*)$, a one-message verification key $\mathsf{VK}_{\mathsf{one}}$, an all-but-one signing key $\mathsf{SK}_{\mathsf{abo}}$ and an all-but-one verification key $\mathsf{VK}_{\mathsf{abo}}$.

$\mathsf{SignSplAbo}(\mathsf{SK}_{\mathsf{abo}}, m)$ The all-but-one signing algorithm is deterministic. It takes as input an all-but-one signing key $\mathsf{SK}_{\mathsf{abo}}$ and a message m, and outputs a signature σ.

Correctness. Let $m^* \in \mathsf{Msg}$ be any message. Let $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\mathsf{rej}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$ and $(\sigma_{\mathsf{one}}, \mathsf{VK}_{\mathsf{one}}, \mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}, m^*)$. Then, we require the following correctness properties:

1. For all $m \in \mathsf{Msg}$, $\mathsf{VerSpl}(\mathsf{VK}, m, \mathsf{SignSpl}(\mathsf{SK}, m)) = 1$.
2. For all $m \in \mathsf{Msg}, m \neq m^*$, $\mathsf{SignSpl}(\mathsf{SK}, m) = \mathsf{SignSplAbo}(\mathsf{SK}_{\mathsf{abo}}, m)$.
3. For all σ, $\mathsf{VerSpl}(\mathsf{VK}_{\mathsf{one}}, m^*, \sigma) = \mathsf{VerSpl}(\mathsf{VK}, m^*, \sigma)$.
4. For all $m \neq m^*$ and σ, $\mathsf{VerSpl}(\mathsf{VK}, m, \sigma) = \mathsf{VerSpl}(\mathsf{VK}_{\mathsf{abo}}, m, \sigma)$.
5. For all $m \neq m^*$ and σ, $\mathsf{VerSpl}(\mathsf{VK}_{\mathsf{one}}, m, \sigma) = 0$.
6. For all σ, $\mathsf{VerSpl}(\mathsf{VK}_{\mathsf{abo}}, m^*, \sigma) = 0$.
7. For all σ and all $m \in \mathsf{Msg}$, $\mathsf{VerSpl}(\mathsf{VK}_{\mathsf{rej}}, m, \sigma) = 0$.

We describe a security property of splittable signatures that will be explicitly used in this work. This notion, termed as $\mathsf{VK}_{\mathsf{one}}$ indistinguishability, states that: given a signature on a message m, an adversary should not be able to distinguish the verification key VK from the split verification key $\mathsf{VK}_{\mathsf{one}}$, that is computed as a result of applying $\mathsf{SplitSpl}$ on the signing key and message m.

We recall the formal definition from KLW, below.

Definition 4 ($\mathsf{VK}_{\mathsf{one}}$ indistinguishability)**.** *A splittable signature scheme* $\mathsf{SplScheme}$ *for a message space* Msg *is said to be* $\mathsf{VK}_{\mathsf{one}}$ *indistinguishable if any PPT adversary* $\mathcal{A}$ *has negligible advantage in the following security game:*

$\mathsf{Expt}(1^\lambda, \mathsf{SplScheme}, \mathcal{A})$*:*

1. *$\mathcal{A}$ sends a message $m^* \in \mathsf{Msg}$.*
2. *Challenger computes $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\mathsf{rej}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$. Next, it computes $(\sigma_{\mathsf{one}}, \mathsf{VK}_{\mathsf{one}}, \mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}, m^*)$. It chooses $b \leftarrow \{0,1\}$. If $b = 0$, it sends $(\sigma_{\mathsf{one}}, \mathsf{VK}_{\mathsf{one}})$ to $\mathcal{A}$. Else, it sends $(\sigma_{\mathsf{one}}, \mathsf{VK})$ to $\mathcal{A}$.*
3. *$\mathcal{A}$ sends its guess b'.*

$\mathcal{A}$ wins if $b = b'$.

We note that in the game above, $\mathcal{A}$ only receives the signature σ_{one} on m^*, on which VK and VK_{one} behave identically.

III. Iterators. The syntax of cryptographic iterators is described below.

Syntax. Let ℓ be any polynomial. An iterator $\mathsf{PP}_{\mathsf{Itr}}$ with message space $\mathsf{Msg}_\lambda = \{0,1\}^{\ell(\lambda)}$ and state space $\mathsf{SplScheme}_\lambda$ consists of three algorithms - $\mathsf{SetupItr}$, $\mathsf{ItrEnforce}$ and $\mathsf{Iterate}$ defined below.

$\mathsf{SetupItr}(1^\lambda, T)$ The setup algorithm takes as input the security parameter λ (in unary), and an integer bound T (in binary) on the number of iterations. It outputs public parameters $\mathsf{PP}_{\mathsf{Itr}}$ and an initial state $v_0 \in \mathsf{SplScheme}_\lambda$.

$\mathsf{ItrEnforce}(1^\lambda, T, \vec{m} = (m_1, \ldots, m_k))$ The enforced setup algorithm takes as input the security parameter λ (in unary), an integer bound T (in binary) and k messages $(m_1, \ldots, m_k)$, where each $m_i \in \{0,1\}^{\ell(\lambda)}$ and k is some polynomial in λ. It outputs public parameters $\mathsf{PP}_{\mathsf{Itr}}$ and a state $v_0 \in \mathsf{SplScheme}$.

$\mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, m)$ The iterate algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Itr}}$, a state v_{in}, and a message $m \in \{0,1\}^{\ell(\lambda)}$. It outputs a state $v_{\mathsf{out}} \in \mathsf{SplScheme}_\lambda$.

Remark 2. *As in the case of positional accumulators, we set T to be 2^λ and not mention T as an explicit input to the above algorithms.*

For simplicity of notation, the dependence of ℓ on λ will not be explicitly mentioned. Also, for any integer $k \leq T$, we will use the notation $\mathsf{Iterate}^k(\mathsf{PP}_{\mathsf{Itr}}, v_0, (m_1, \ldots, m_k))$ to denote $\mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{k-1}, m_k)$, where $v_j = \mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{j-1}, m_j)$ for all $1 \leq j \leq k-1$.

3 1-Key ABE for TMs with Additive Overhead

We adapt the definition of attribute based encryption to the case when the keys correspond to Turing machines as against circuits. Further, there is no a priori bound placed on either the lengths of the attributes or the messages. Recall that in a attribute based encryption scheme, every ciphertext of a message m is associated to an attribute x such that an evaluator holding a key of a predicate P can recover m if and only if $P(x) = 1$.

The concept of ABE for TMs was first studied in the work by Goldwasser et al. [27]. Although unlike Goldwasser et al., we further restrict this to the setting where the adversary only makes a single key query. We call this a *single-key* (or 1-key) ABE scheme for Turing machines. A formal definition of this primitive is provided in Appendix B.

1-Key Attribute Based Encryption for TMs with Additive Overhead. We can additionally enforce that the 1-key attribute based encryption for TMs scheme satisfies the following condition: the size of the ABE key of M is essentially $|M|$, up to an additive factor of polynomial in the security factor. We term such a notion to be a 1-key attribute based encryption for TMs scheme with additive overhead. More formally,

Definition 5. *A 1-key attribute based encryption for TMs scheme,* $\mathsf{1ABE}$*, defined for a class of Turing machines* $\mathcal{M}$*, is said to have additive overhead if* $|\mathsf{1ABE}.sk_M| = |M| + \text{poly}(\lambda)$*, where* $(\mathsf{1ABE.SK}, \mathsf{1ABE.PP}) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$ *and* $\mathsf{1ABE}.sk_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M \in \mathcal{M})$*.*

3.1 Construction

There are three main tools that are central to our construction of 1-key ABE for TMs with additive overhead, namely accumulators, splittable signatures scheme and iterators. These primitives are imported from the work of Koppula et al. We give a brief description of the three primitives below. The formal treatment of these primitives are provided in Section 2.

1. **Storage accumulators**: This is a device used to store authenticated data. It is associated with a mechanism to update the data along with its authentication. The authenticated value is short and in particular independent of the size of the storage. Using the authenticated value it can be checked whether a block of storage is "valid" – validity here refers to the fact that the storage is computed as per the scheme. A key property of this primitive is that the parameters can be programmed in such a way that there does not exist any false proof to validate a maliciously computed piece of storage.

   We use the specific accumulator scheme, based on iO and one-way functions, designed by KLW in our construction of 1-key ABE. The accumulators scheme that will be used in the construction of 1-key ABE for TMs is denoted by $\mathsf{Acc} = (\mathsf{SetupAcc}, \mathsf{EnforceRead}, \mathsf{EnforceWrite}, \mathsf{PrepRead}, \mathsf{PrepWrite}, \mathsf{VerifyRead}, \mathsf{WriteStore}, \mathsf{Update})$. It is associated with the message space Σ_{tape} with accumulated value of size ℓ_{Acc} bits.

2. **Iterators**: It allows for maintaining state information that is regularly updated based on the sequence of messages it receives. The security property allows for programming the parameters such that there exists a unique sequence of messages that leads to a particular state value.

   The iterators scheme is denoted by $\mathsf{Itr} =(\mathsf{SetupItr}, \mathsf{ItrEnforce}, \mathsf{Iterate})$. It is associated with the message space $\{0,1\}^{2\lambda+\ell_{\mathsf{Acc}}}$ with iterated value of size ℓ_{Itr} bits.

3. **Splittable signatures**: As the name suggests, it is a type of a signature scheme that allows for splitting the signing key-verification key pair into two pairs of signature-verification keys such that each pair acts upon a unique partition of the message space. That is, a pair $(\mathsf{SK}, \mathsf{VK})$ can be split into two pairs $(\mathsf{SK}_{\mathcal{U}}, \mathsf{VK}_{\mathcal{U}})$ and $(\mathsf{SK}_{\mathcal{V}}, \mathsf{VK}_{\mathcal{V}})$, where (i) $\mathcal{U}$ and $\mathcal{V}$ form a partition of the message space, and (ii) $(\mathsf{SK}_{\mathcal{U}}, \mathsf{VK}_{\mathcal{U}})$ (resp., $(\mathsf{SK}_{\mathcal{V}}, \mathsf{VK}_{\mathcal{V}})$) is a signing key-verification key pair that is used to validate only messages in $\mathcal{U}$ (resp., $\mathcal{V}$). Koppula et al. consider the case when one of the partition contains just one element. We work in the same setting as Koppula et al.

   We denote the splittable signatures scheme by $\mathsf{SplScheme} = (\mathsf{SetupSpl}, \mathsf{SignSpl}, \mathsf{VerSpl}, \mathsf{SplitSpl}, \mathsf{SignSplAbo})$. It is associated with the message space $\{0,1\}^{\ell_{\mathsf{Itr}}+\ell_{\mathsf{Acc}}+2\lambda}$.

In addition, we also use a puncturable PRF family denoted by F.

We now describe the scheme $\mathsf{1ABE} = (\mathsf{1ABE.Setup}, \mathsf{1ABE.KeyGen}, \mathsf{1ABE.Enc}, \mathsf{1ABE.Dec})$ below. We use the message hiding encodings construction of KLW as the basic template for our scheme. Let the scheme $\mathsf{1ABE}$ be associated to the class of Turing machines $\mathcal{M}$. Without loss of generality, the start state of every Turing machine in $\mathcal{M}$ is denoted by q_0. We denote the message space to be MSG.

$\underline{\mathsf{1ABE.Setup}(1^\lambda)}$: On input security parameter λ, it first executes the setup of splittable signatures scheme, $(\mathsf{SK}_{\mathsf{tm}}, \mathsf{VK}_{\mathsf{tm}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$. It then executes the setup of the accumulator setup to obtain the values, $(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{SetupAcc}(1^\lambda)$. It then executes the setup of the iterator scheme to obtain the public parameters, $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{SetupItr}(1^\lambda)$.

It finally outputs the following public key-secret key pair,

$$\Big(\mathsf{1ABE.PP} = (\mathsf{VK_{tm}}, \mathsf{PP_{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP_{Itr}}, v_0), \mathsf{1ABE.SK} = (\mathsf{1ABE.PP}, \mathsf{SK_{tm}})\Big)$$

<u>$\mathsf{1ABE.KeyGen}(\mathsf{SK_{tm}}, M \in \mathcal{M})$</u>: On input the master secret key $\mathsf{1ABE.SK} = (\mathsf{1ABE.PP}, \mathsf{SK_{tm}})$ and $M \in \mathcal{M}$, it executes the following steps:

1. It parses the public key $\mathsf{1ABE.PP}$ as $(\mathsf{VK_{tm}}, \mathsf{PP_{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP_{Itr}}, v_0)$.
2. **Initialization of the storage tree**: Let $\ell_{\mathsf{tm}} = |M|$ be the length of the machine M. For $1 \leq j \leq \ell_{\mathsf{tm}}$, it computes $\widetilde{store}_j = \mathsf{WriteStore}(\mathsf{PP_{Acc}}, \widetilde{store}_{j-1}, j-1, M_j)$, $aux_j = \mathsf{PrepWrite}(\mathsf{PP_{Acc}}, \widetilde{store}_{j-1}, j-1)$, $\widetilde{w}_j = \mathsf{Update}(\mathsf{PP_{Acc}}, \widetilde{w}_{j-1}, M_j, j-1, aux_j)$, where M_j denotes the j^{th} bit of M. Finally, it sets the root $w_0 = \widetilde{w}_{\ell_{\mathsf{tm}}}$.
3. **Signing the accumulator value**: It generates the signature on the message $(v_0, q_0, w_0, 0)$, $\sigma_0 \leftarrow \mathsf{SignSpl}(\mathsf{SK_{tm}}, \mu = (v_0, q_0, w_0, 0))$, where q_0 is the start state of M.

It outputs the ABE key $\mathsf{1ABE.}sk_M = (M, w_0, \sigma_{\mathsf{tm}}, v_0)$.

[Note: The key generation does not output the storage tree $store_0$ but instead it just outputs the initial store value $\widetilde{store}_0$. The evaluator in possession of M, $\widetilde{store}_0$ and $\mathsf{PP_{Acc}}$ can reconstruct the tree $store_0$.]

<u>$\mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg})$</u>: On input the public key $\mathsf{1ABE.PP} = (\mathsf{VK_{tm}}, \mathsf{PP_{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP_{Itr}}, v_0)$, attribute $x \in \{0,1\}^*$ and message $\mathsf{msg} \in \mathsf{MSG}$, it executes the following steps:

1. It first samples a PRF key K_A at random from the family F.
2. **Obfuscating the next step function**: Consider a universal Turing machine $U_x(\cdot)$ that on input M executes M on x for at most 2^λ steps and outputs $M(x)$ if M terminates, otherwise it outputs $\perp$. It computes the obfuscation of the program NxtMsg in 1, namely $N \leftarrow i\mathcal{O}(\mathsf{NxtMsg}\{U_x(\cdot), \mathsf{msg}, \mathsf{PP_{Acc}}, \mathsf{PP_{Itr}}, K_A\})$. At its core, NxtMsg is essentially the next message function of the Turing machine $U_x(\cdot)$ – it takes as input a TM M and outputs $M(x)$ if it halts within 2^λ else it outputs $\perp$. In addition, it performs checks to validate whether the previous step was correctly computed. It also generates authentication values for the current step.
3. It computes the obfuscation of the program $S \leftarrow (\mathsf{SignProg}\{K_A, \mathsf{VK_{tm}}\})$ where $\mathsf{SignProg}$ is defined in Figure 2. The program $\mathsf{SignProg}$ takes as input a message-signature pair and outputs a signature with respect to a different key on the same message.

It outputs the ciphertext, $\mathsf{1ABE.CT} = (N, S)$.

<u>$\mathsf{1ABE.Dec}(\mathsf{1ABE.}sk_M, \mathsf{1ABE.CT})$</u>: On input the ABE key $\mathsf{1ABE.}sk_M = (M, w_0, \sigma_{\mathsf{tm}}, v_0)$ and ciphertext $\mathsf{1ABE.CT} = (N, S)$, it first executes the obfuscated program $S\big(y = (v_0, q_0, w_0, 0), \sigma_{\mathsf{tm}}\big)$ to obtain σ_0. It then executes the following steps.

1. **Reconstructing the storage tree**: Suppose $\ell_{\mathsf{tm}} = |M|$ be the length of the TM M. For $1 \leq j \leq \ell_{\mathsf{tm}}$, it then repeatedly updates the storage tree by computing, $\widetilde{store}_j = \mathsf{WriteStore}(\mathsf{PP_{Acc}}, \widetilde{store}_{j-1}, j-1, M_j)$. Finally, it sets $store_0 = \widetilde{store}_{\ell_{\mathsf{tm}}}$.
2. **Executing N one step at a time**: For $i = 1$ to 2^λ,
   (a) Compute the proof that validates the storage value $store_{i-1}$ (storage value at $(i-1)^{th}$ time step) at position pos_{i-1}. Let $(\mathsf{sym}_{i-1}, \pi_{i-1}) \leftarrow \mathsf{PrepRead}(\mathsf{PP_{Acc}}, store_{i-1}, \mathsf{pos}_{i-1})$.
   (b) Compute the auxiliary value, $aux_{i-1} \leftarrow \mathsf{PrepWrite}(\mathsf{PP_{Acc}}, store_{-1}, \mathsf{pos}_{i-1})$.
   (c) Run the obfuscated next message function. Compute $\mathsf{out} \leftarrow N(i, \mathsf{sym}_{i-1}, \mathsf{pos}_{i-1}, \mathsf{st}_{i-1}, w_{i-1}, v_{i-1}, \sigma_{i-1}, \pi_{i-1}, aux_{i-1})$. If $\mathsf{out} \in \mathsf{MSG} \cup \{\perp\}$. output out.
   Else parse out as $(\mathsf{sym}_{w,i}, \mathsf{pos}_i, \mathsf{st}_i, w_i, v_i, \sigma_i)$.
   (d) Compute the storage value, $store_i \leftarrow \mathsf{WriteStore}(\mathsf{PP_{Acc}}, store_{i-1}, \mathsf{pos}_{i-1}, \mathsf{sym}_{w,i})$.

```
Program $\mathsf{NxtMsg}$

Constants: Turing machine $U_x = \langle Q, \Sigma_{\text{tape}}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, message $\mathsf{msg}$, Public parameters for accumulator $\mathsf{PP}_{\mathsf{Acc}}$, Public parameters for Iterator $\mathsf{PP}_{\mathsf{Itr}}$, Puncturable PRF key $K_A \in \mathcal{K}$.

Input: Time $t \in [T]$, symbol $\mathsf{sym}_{\mathsf{in}} \in \Sigma_{\text{tape}}$, position $\mathsf{pos}_{\mathsf{in}} \in [T]$, state $\mathsf{st}_{\mathsf{in}} \in Q$, accumulator value $w_{\mathsf{in}} \in \{0,1\}^{\ell_{\mathsf{Acc}}}$, Iterator value $v_{\mathsf{in}}$, signature $\sigma_{\mathsf{in}}$, accumulator proof $\pi$, auxiliary value $aux$.

1. Verification of the accumulator proof:
    • If $\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_{\mathsf{in}}, \mathsf{sym}_{\mathsf{in}}, \mathsf{pos}_{\mathsf{in}}, \pi) = 0$ output $\perp$.
2. Verification of signature on the input state, position, accumulator and iterator values:
    • Let $F(K_A, t-1) = r_A$. Compute $(\mathsf{SK}_A, \mathsf{VK}_A, \mathsf{VK}_{A,\mathsf{rej}}) = \mathsf{SetupSpl}(1^\lambda; r_A)$.
    • Let $m_{\text{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\mathsf{in}}, w_{\mathsf{in}}, \mathsf{pos}_{\mathsf{in}})$. If $\mathsf{VerSpl}(\mathsf{VK}_A, m_{\text{in}}, \sigma_{\mathsf{in}}) = 0$ output $\perp$.
3. Executing the transition function:
    • Let $(\mathsf{st}_{\mathsf{out}}, \mathsf{sym}_{\mathsf{out}}, \beta) = \delta(\mathsf{st}_{\mathsf{in}}, \mathsf{sym}_{\mathsf{in}})$ and $\mathsf{pos}_{\mathsf{out}} = \mathsf{pos}_{\mathsf{in}} + \beta$.
    • If $\mathsf{st}_{\mathsf{out}} = q_{\text{rej}}$ output $\perp$.
    • If $\mathsf{st}_{\mathsf{out}} = q_{\text{acc}}$ output $\mathsf{msg}$.
4. Updating the accumulator and the iterator values:
    • Compute $w_{\mathsf{out}} = \mathsf{Accumulate}(\mathsf{PP}_{\mathsf{Acc}}, w_{\mathsf{in}}, \mathsf{sym}_{\mathsf{out}}, \mathsf{pos}_{\mathsf{in}}, aux)$. If $w_{\mathsf{out}} = Reject$, output $\perp$.
    • Compute $v_{\mathsf{out}} = \mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, (\mathsf{st}_{\mathsf{in}}, w_{\mathsf{in}}, \mathsf{pos}_{\mathsf{in}}))$.
5. Generating the signature on the new state, position, accumulator and iterator values:
    • Let $F(K_A, t) = r'_A$. Compute $(\mathsf{SK}'_A, \mathsf{VK}'_A, \mathsf{VK}'_{A,\mathsf{rej}}) \leftarrow \mathsf{SetupSpl}(1^\lambda; r'_A)$.
    • Let $m_{\text{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\mathsf{out}}, w_{\mathsf{out}}, \mathsf{pos}_{\mathsf{out}})$ and $\sigma_{\mathsf{out}} = \mathsf{SignSpl}(\mathsf{SK}'_A, m_{\text{out}})$.
6. Output $\mathsf{sym}_{\mathsf{out}}, \mathsf{pos}_{\mathsf{out}}, \mathsf{st}_{\mathsf{out}}, w_{\mathsf{out}}, v_{\mathsf{out}}, \sigma_{\mathsf{out}}$.
```

Figure 1: Program NxtMsg

```
Program $\mathsf{SignProg}$

Constants: PRF key $K_A$ and verification key $\mathsf{VK}_{\mathsf{tm}}$.
Input: Message $y$ and a signature $\sigma_{\mathsf{tm}}$.

1. If $\mathsf{VerSpl}(\mathsf{VK}_{\mathsf{tm}}, y, \sigma_{\mathsf{tm}}) = 0$ then output $\perp$.
2. Execute the pseudorandom function on input 0 to obtain $r_A \leftarrow F(K, 0)$. Generate the setup of splittable signatures scheme, $(\mathsf{SK}_0, \mathsf{VK}_0) \leftarrow \mathsf{SetupSpl}(1^\lambda; r_A)$.
3. Compute the signature $\sigma_0 \leftarrow \mathsf{SignSpl}(\mathsf{SK}_0, y)$.
4. Output $\sigma_0$.
```

Figure 2: Program $\mathsf{SignProg}$

This completes the description of the scheme. The correctness of the above scheme follows along the same lines as the correctness of the message hiding scheme of Koppula et al. For completeness, we give a proof sketch below.

Lemma 1. $\mathsf{1ABE}$ *satisfies the correctness property of an ABE scheme.*

Proof sketch. Suppose $\mathsf{1ABE.CT}$ is a ciphertext of message msg w.r.t attribute x and $\mathsf{1ABE}.sk_M$ is an ABE key of machine M. We claim that in the i^{th} iteration of the decryption of $\mathsf{1ABE.CT}$ using $\mathsf{1ABE}.sk_M$, the storage corresponds to the work tape of the execution of $M(x)$ at the i^{th} time step, denoted by $W_{t=i}$ [10]. Once we show this, the lemma

[10]To be more precise, the storage in the KLW construction is a tree with the j^{th} leaf containing the value of the j^{th} location in the work tape $W_{t=i}$.

follows.

We prove this claim by induction on the total number of steps in the TM execution. The base case corresponds to 0^{th} time step when the iterations haven't begun. At this point, the storage corresponds to the description of the machine M which is exactly $W_{t=0}$ (work tape at time step 0). In the induction hypothesis, we assume that at time step $i-1$, the storage contains the work tape $W_{t=i-1}$. We need to argue for the case when $t = i$. To take care of this case, we just need to argue that the obfuscated next step function computes the i^{th} step of the execution of $M(x)$ correctly. The correctness of obfuscated next step function in turn follows from the correctness of iO and other underlying primitives.

Remark 3. *In the description of Koppula et al., the accumulator and the iterator algorithms also took the time bound T as input. Here, we set $T = 2^{\lambda}$ since we are only concerned with Turing machines that run in time polynomial in λ.*

Additive overhead. Suppose $\mathsf{1ABE}.sk_M = (M, w_0, \widetilde{store}_0, \sigma_{\mathsf{tm}}, v_0)$ be the ABE key generated as the output of $\mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M \in \mathcal{M})$. From the efficiency property of accumulators, we have $|w_0|$ and $|\widetilde{store}_0|$ to be just polynomials in the security parameter λ. The signature σ_{tm} on the message w_0 is also a polynomial in the security parameter. Lastly, the iterator parameter v_0 is also a polynomial in the security parameter. Thus, the size of $\mathsf{1ABE}.sk_M$ is $|M| + \mathrm{poly}(\lambda)$.

3.2 Security

To prove the security of our scheme we make use of a theorem proved in Koppula et al. Before we recall their theorem, we first define the following distribution that would be useful to state the theorem. This distribution is identical to the output distribution of input encoding of the message hiding encoding scheme by [33]. We denote the distribution by $\mathcal{D}_{M,U_x(\cdot),\mathsf{msg}}$, where M is a Turing machine, $x \in \{0,1\}^*$ and $\mathsf{msg} \in \mathsf{MSG}$. We define the sampler for the distribution below. We use the same notation to denote both the distribution as well as its sampler.

$\underline{\mathcal{D}_{M,x,\mathsf{msg}}(1^{\lambda})}$: It first computes $(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{SetupAcc}(1^{\lambda}, T)$. Let $\ell_{\mathsf{tm}} = |M|$ be the length of the Turing machine M. It computes $\widetilde{store}_j = \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1, M_j)$, $aux_j = \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1)$, $\widetilde{w}_j = \mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_{j-1}, \mathsf{inp}_j, j-1, aux_j)$ for $1 \leq j \leq \ell_{\mathsf{tm}}$. Finally, it sets $w_0 = \widetilde{w}_{\ell_{\mathsf{tm}}}$ and $s_0 = \widetilde{store}_{\ell_{\mathsf{tm}}}$. Next, it computes the iterator parameters $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{SetupItr}(1^{\lambda}, T)$. It chooses a puncturable PRF key $K_A \leftarrow F.\mathsf{Setup}(1^{\lambda})$. It also computes an obfuscation $N \leftarrow i\mathcal{O}(\mathsf{NxtMsg}\{U_x(\cdot), \mathsf{msg}, \mathsf{PP}_{\mathsf{Acc}}, \mathsf{PP}_{\mathsf{Itr}}, K_A\})$ where NxtMsg is defined in Figure 1. Let $r_A = F(K_A, 0)$, $(\mathsf{SK}_0, \mathsf{VK}_0) = \mathsf{SetupSpl}(1^{\lambda}; r_A)$ and $\sigma_0 = \mathsf{SignSpl}(\mathsf{SK}_0, (v_0, q_0, w_0, 0))$.

The distribution finally outputs the following:

$$\left(N, w_0, v_0, \sigma_0, store_0, \mathsf{init} = (\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}})\right)$$

[*Remark: The values $\widetilde{w}_0$, widetildestore$_0$ and* $\mathsf{PP}_{\mathsf{Itr}}$ *are not explicitly given out in the message hiding encodings construction of KLW. But in their specific accumulator construction (which even we are utilizing), $\widetilde{w}_0$ is set to be $\perp$ and $\widetilde{store}_0$ is set to be $\perp$. Although not made explicit, even the iterator public parameters* $\mathsf{PP}_{\mathsf{Itr}}$ *can be given out in their construction without any modification in the proof of security.*]

The following theorem was shown in [33].

Theorem 2 ([33],Theorem 6.1)**.** *For all TMs $M \in \mathcal{M}$, $x \in \{0,1\}^*, \mathsf{msg}_0, \mathsf{msg}_1 \in \mathsf{MSG}$ such that $M(x) = 0$ and $|\mathsf{msg}_0| = |\mathsf{msg}_1|$, we have that the distributions $\mathcal{D}_{M,x,\mathsf{msg}_0}$ and $\mathcal{D}_{M,x,\mathsf{msg}_1}$ are computationally indistinguishable assuming the security of indistinguishability obfuscators* $i\mathcal{O}$*, accumulators scheme* Acc*, iterators scheme* Itr*, splittable signatures scheme* SplScheme*.*

We prove the following theorem in Appendix C.

Theorem 3. *The scheme* $\mathsf{1ABE}$ *for the class of Turing machines* $\mathcal{M}$*, is weak-selectively secure assuming the security of indistinguishability obfuscators* $\mathsf{i}\mathcal{O}$*, accumulators scheme* Acc*, iterators scheme* Itr *and splittable signatures scheme* $\mathsf{SplScheme}$*.*

Since the accumulators, iterators and splittable signatures can be instantiated from iO and one way functions, we have the following corollary.

Corollary 1. *There exists a 1-key ABE for TMs scheme assuming the existence of indistinguishability obfuscators for P/poly and one-way functions.*

3.3 1-Key Two-Outcome ABE for TMs

Goldwasser et al. [27] proposed the notion of 1-key two-outcome ABE for circuits as a variant of 1-key attribute based encryption for circuits where a pair of secret messages are encoded as against just one secret message. Depending on the output of the predicate, exactly one of the messages is revealed and the other message is hidden. That is, the encryption is performed on a single attribute x and two messages $(\mathsf{msg}_0, \mathsf{msg}_1)$. The decryption on input an ABE key $\mathsf{TwoMsgABE}.sk_M$ and ciphertext $\mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)}$ outputs msg_0 if $M(x) = 0$ and outputs msg_1 if $M(x) = 1$. The security guarantee then says that if $M(x) = 0$ (resp., $M(x) = 1$) then the pair $(\mathsf{TwoMsgABE}.sk_M, \mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)})$, reveal no information about msg_1 (resp., msg_0).

We adopt their definition but in the case when the predicates are implemented as Turing machines instead of circuits. We give a formal definition and a simple construction of this primitive in Appendix D.

4 Oblivious Evaluation Encodings

The main building block in our construction of iO with constant multiplicative overhead is the notion of *oblivious evaluation encodings* (OEE). This is a strengthening of the notion of machine hiding encodings (MHE) introduced in [33]. Recall that machine hiding encodings are essentially randomized encodings (RE) for Turing machines, except the fact that in MHE, the machine needs to be hidden whereas in RE, the input needs to be hidden. That is, a MHE scheme has an encoding procedure that encodes the output of a Turing machine M and an input x. The encoding procedure is much "simpler" than actually computing M on x. There is a decode procedure that decodes the output $M(x)$. The security guarantee now states that the encoding does not reveal anything more than $M(x)$. We make several changes to this definition to obtain our definition of OEE.

Firstly, we encode the machine and the input separately. Secondly, the machine encoding takes as input two Turing machines (M_0, M_1) and outputs a joint (or dual) encoding. Correspondingly, the input encoding now also takes as input a bit b in addition to the actual input x, where b indicates which of the two machines M_0 or M_1 needs to be used. In terms of security, we require the following two properties to be satisfied:

- Any PPT adversary should not be able to distinguish encodings of (M_0, M_0) and (M_0, M_1) (resp., (M_1, M_1) and (M_0, M_1)) even if the adversary is given a *punctured* input encoding key that allows him to encode inputs of the form $(x, 0)$ (resp., $(x, 1)$).
- Any PPT adversary is unable to distinguish the encodings of $(x, 0)$ and $(x, 1)$ even given an oblivious evaluation encoding (M_0, M_1), where $M_0(x) = M_1(x)$ and another type of punctured input encoding key that allows him to generate input encodings of $(x', 0)$ and $(x', 1)$ for all $x' \neq x$.

4.1 Definition

Syntax. We describe the syntax of a oblivious evaluation encoding scheme OEE below. The class of Turing machines associated with the scheme is $\mathcal{M}$ and the input space is $\{0, 1\}^*$.

Although we consider inputs of arbitrary lengths, during the generation of the parameters we place an upper bound on the running time of the machines which automatically puts an upper bound on the length of the inputs.

- $\mathsf{OEE.Setup}(1^\lambda)$: It takes as input a security parameter λ and outputs a secret key $\mathsf{OEE.sk}$.
- $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$: It takes as input a secret key $\mathsf{OEE.sk}$, a pair of Turing machines $M_0, M_1 \in \mathcal{M}$ and outputs a joint encoding $(\widetilde{M_0, M_1})$.
- $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$: It takes as input a secret key $\mathsf{OEE.sk}$, an input $x \in \{0,1\}^*$, a choice bit b and outputs an input encoding $(\widetilde{x, b})$.
- $\mathsf{OEE.Decode}((\widetilde{M_0, M_1}), (\widetilde{x, b}))$: It takes as input a joint Turing machine encoding $(\widetilde{M_0, M_1})$, an input encoding $(\widetilde{x, b})$, and outputs a value z.

In addition to the above main algorithms, there are four helper algorithms.

- $\mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x)$: It takes as input a secret key $\mathsf{OEE.sk}$, input $x \in \{0,1\}^*$ and outputs a punctured key $\mathsf{OEE.sk}_x$.
- $\mathsf{OEE.pIEncode}(\mathsf{OEE.sk}_x, x', b)$: It takes as input a punctured secret key $\mathsf{OEE.sk}_x$, an input $x' \neq x$, a bit b and outputs an input encoding $(\widetilde{x', b})$.
- $\mathsf{OEE.puncBit}(\mathsf{OEE.sk}, b)$: It takes as input a secret key $\mathsf{OEE.sk}$, an input bit b and outputs a key $\mathsf{OEE}.sk_b$.
- $\mathsf{OEE.pBEncode}(\mathsf{OEE}.sk_b, x)$: It takes as input a key $\mathsf{OEE}.sk_b$, an input x and outputs an input encoding $(\widetilde{x, b})$.

Correctness. We say that an OEE scheme is correct if it satisfies the following three properties:

1. *Correctness of Encode and Decode:* For all $M_0, M_1 \in \mathcal{M}$, $x \in \{0,1\}^*$ and $b \in \{0,1\}$,
$$\mathsf{OEE.Decode}(\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1),$$
$$\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)) = M_b(x),$$
where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$.
2. *Correctness of Input Puncturing:* For all $M_0, M_1 \in \mathcal{M}$, $x, x' \in \{0,1\}^*$ such that $x' \neq x$ and $b \in \{0,1\}$,
$$\mathsf{OEE.Decode}\left((\widetilde{M_0, M_1}), (\widetilde{x', b})\right) = M_b(x'),$$
where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}\left(1^\lambda\right)$; $(\widetilde{M_0, M_1}) \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ and $(\widetilde{x', b}) \leftarrow \mathsf{OEE.pIEncode}(\mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x), x', b)$.
3. *Correctness of Bit Puncturing:* For all $M_0, M_1 \in \mathcal{M}$, $x \in \{0,1\}^*$ and $b \in \{0,1\}$,
$$\mathsf{OEE.Decode}\left((\widetilde{M_0, M_1}), (\widetilde{x, b})\right) = M_b(x),$$
where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}\left(1^\lambda\right)$, $(\widetilde{M_0, M_1}) \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ and $(\widetilde{x, b}) \leftarrow \mathsf{OEE.pBEncode}\left(\mathsf{OEE.puncBit}\left(\mathsf{OEE.sk}, b\right), x\right)$.

Efficiency. We require that an OEE scheme satisfies the following efficiency conditions. Informally, we require that the Turing machine encoding (resp., input encoding) algorithm only has a logarithmic dependence on the time bound. Furthermore, the running time of the decode algorithm should take time proportional to the computation time of the encoded Turing machine on the encoded input.

1. The running time of $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0 \in \mathcal{M}, M_1 \in \mathcal{M})$ is a polynomial in $(\lambda, |M_0|, |M_1|)$, where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$.

2. The running time of $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x \in \{0,1\}^*, b)$ is a polynomial in $(\lambda, |x|)$, where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$.
3. The running time of $\mathsf{OEE.Decode}(\widetilde{(M_0, M_1)}, \widetilde{(x,b)})$ is a polynomial in $(\lambda, |M_0|, |M_1|, |x|, t)$, where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$, $\widetilde{(M_0, M_1)} \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0 \in \mathcal{M}, M_1 \in \mathcal{M})$, $\widetilde{(x,b)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x \in \{0,1\}^*, b)$ and t is the running time of the Turing machine M_b on x.

Indistinguishability of Encoding Bit. We describe security of encoding bit as a multi-stage game between an adversary $\mathcal{A}$ and a challenger.

- Setup: $\mathcal{A}$ chooses two Turing machines $M_0, M_1 \in \mathcal{M}$ and an input x such that $|M_0| = |M_1|$ and $M_0(x) = M_1(x)$. $\mathcal{A}$ sends the tuple (M_0, M_1, x) to the challenger.

  The challenger chooses a bit $b \in \{0,1\}$ and computes the following: (a) $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$, (b) machine encoding $\widetilde{(M_0, M_1)} \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$, (c) input encoding $\widetilde{(x,b)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$, and (d) punctured key $\mathsf{OEE.sk}_x \leftarrow \mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x)$. Finally, it sends the following tuple to $\mathcal{A}$:

$$\left(\widetilde{(M_0, M_1)}, \widetilde{(x,b)}, \mathsf{OEE.sk}_x\right).$$

- Guess: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

The advantage of $\mathcal{A}$ in this game is defined as $\mathsf{adv}_{\mathsf{PMHE}} = \Pr[b' = b] - \frac{1}{2}$.

Definition 6 (Indistinguishability of encoding bit)**.** *An OEE scheme satisfies indistinguishability of encoding bit if there exists a neglible function* $\mathsf{negl}(\cdot)$ *such that for every PPT adversary* $\mathcal{A}$ *in the above security game,* $\mathsf{adv}_{\mathsf{PMHE}} = \mathsf{negl}(\lambda)$.

Indistinguishability of Machine Encoding. We describe security of machine encoding as a multi-stage game between an adversary $\mathcal{A}$ and a challenger.

- Setup: $\mathcal{A}$ chooses two Turing machines $M_0, M_1 \in \mathcal{M}$ and a bit $c \in \{0,1\}$ such that $|M_0| = |M_1|$. $\mathcal{A}$ sends the tuple (M_0, M_1, c) to the challenger.

  The challenger chooses a bit $b \in \{0,1\}$ and computes the following: (a) $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$, (b) $\widetilde{(\mathsf{TM}_1, \mathsf{TM}_2)} \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, \mathsf{TM}_1, \mathsf{TM}_2)$, where $\mathsf{TM}_1 = M_0, \mathsf{TM}_2 = M_{1 \oplus b}$ if $c = 0$ and $\mathsf{TM}_1 = M_{0 \oplus b}, \mathsf{TM}_2 = M_1$ otherwise, and (c) $\mathsf{OEE.sk}_b \leftarrow \mathsf{OEE.puncBit}(\mathsf{OEE.sk}, c)$. Finally, it sends the following tuple to $\mathcal{A}$:

$$\left(\widetilde{(\mathsf{TM}_1, \mathsf{TM}_2)}, \mathsf{OEE.sk}_c\right).$$

- Guess: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

The advantage of $\mathcal{A}$ in this game is defined as $\mathsf{adv} = \Pr[b' = b] - \frac{1}{2}$.

Definition 7 (Indistinguishability of machine encoding)**.** *An OEE scheme satisfies indistinguishability of machine encoding if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every PPT adversary* $\mathcal{A}$ *in the above security game,* $\mathsf{adv}_{\mathrm{PMHE}_2} = \mathsf{negl}(\lambda)$.

OEE with Constant Multiplicative Overhead. The efficiency property in OEE dictates that the output length of the Turing machine encoding algorithm is a polynomial in the size of the Turing machine. We can restrict this condition further by requiring that the Turing machine encoding is only *linear* in the Turing machine size. We term the notion of OEE that satisfies this property as *OEE with constant multiplicative overhead.*

Definition 8 (OEE with constant multiplicative overhead)**.** *An oblivious evaluation encoding scheme for a class of Turing machines* $\mathcal{M}$ *is said to have constant multiplicative overhead if its Turing machine encoding algorithm* $\mathsf{OEE.TMEncode}$ *on input* $(\mathsf{OEE.sk}, M_0, M_1)$ *outputs an encoding* $\widetilde{(M_0, M_1)}$ *such that* $|\widetilde{(M_0, M_1)}| = c \cdot (|M_0| + |M_1|) + \mathrm{poly}(\lambda)$*, where c is a constant* > 0.

4.2 Construction

To construct a oblivious evaluation encoding scheme, we require the following ingredients. We denote the class of Turing machines associated with oblivious evaluation encoding to be $\mathcal{M}$. We make the following notational simplifications: $\mathcal{M}$ consists of only single-bit output Turing machines. In every machine M in $\mathcal{M}$, there is a special location on the worktape in which the output of the Turing machine (0 or 1) is written. Until the termination of the Turing machine, this location contains the symbol $\perp$. We use the notation $M(x)$ to denote the value contained in this special location.

1. A 1-key two-outcome ABE for TMs scheme defined for a class of Turing machines $\mathcal{M}$, represented by $(\mathsf{TwoMsgABE.Setup}, \mathsf{TwoMsgABE.TMEncode}, \mathsf{TwoMsgABE.InpEncode}, \mathsf{TwoMsgABE.Decode})$.
2. A fully homomorphic encryption scheme for circuits with *additive overhead* (Section 2), represented by $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$.
3. A garbling scheme GC, represented by $\mathsf{GC} = (\mathsf{Garble}, \mathsf{EvalGC})$.

We denote the oblivious evaluation encoding scheme to be $\mathsf{OEE} = (\mathsf{OEE.Setup}, \mathsf{OEE.InpEncode}, \mathsf{OEE.TMEncode}, \mathsf{OEE.Decode})$. We denote the additional auxiliary algorithms to be $(\mathsf{OEE.puncInp}, \mathsf{OEE.pIEncode}, \mathsf{OEE.puncBit}, \mathsf{OEE.pBEncode})$. The construction of OEE is presented below.

$\underline{\mathsf{OEE.Setup}(1^\lambda)}$: On input a security parameter λ in unary, it executes the following steps.

- It runs $\mathsf{TwoMsgABE.Setup}(1^\lambda)$ to obtain the secret key-public parameters pair, $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP})$.
- It runs $\mathsf{FHE.Setup}(1^\lambda)$ twice to obtain the FHE public key-secret key pairs $(\mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0)$ and $(\mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.

It finally outputs $\mathsf{OEE.sk} = (\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.

$\underline{\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)}$: On input the secret key $\mathsf{OEE.sk}$ and a pair of Turing machines $M_0, M_1 \in \mathcal{M}$, it does the following.

- It parses $\mathsf{OEE.sk}$ as $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.
- It generates the FHE ciphertexts of TMs M_0 and M_1 w.r.t public keys $\mathsf{FHE.pk}_0$ and $\mathsf{FHE.pk}_1$. That is, it generates $\mathsf{FHE.CT}_{M_0} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_0, M_0)$ and $\mathsf{FHE.CT}_{M_1} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_1, M_1)$.
- It computes the TM encoding of the machine $N = N_{\left(\{\mathsf{FHE.pk}_b, \mathsf{FHE.CT}_{M_b}\}_{b\in\{0,1\}}\right)}$, $\widetilde{N} \leftarrow \mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, N)$, where N is described in Figure 3.

It outputs the TM encoding, $\widetilde{(M_0, M_1)} = \widetilde{N}$.

$$N_{\left(\{\mathsf{FHE.pk}_b, \mathsf{FHE.CT}_{M_b}\}_{b\in\{0,1\}}\right)}(x, i, \mathsf{ind}_t)$$

- Let $U = U_{x,\mathsf{ind}_t}(\cdot)$ be a universal Turing machine that on input a Turing machine M, outputs $M(x)$ if the computation terminates within 2^{ind_t} number of steps, otherwise it outputs $\perp$ [11].
- Transform the universal Turing machine U into a circuit using Theorem 5 (Section 2) by computing $C \leftarrow \mathsf{TMtoCKT}(U)$.
- Execute $\mathsf{FHE.Eval}(\mathsf{FHE.pk}_0, C, \mathsf{FHE.CT}_{M_0})$ to obtain z_1. Similarly execute $\mathsf{FHE.Eval}(\mathsf{FHE.pk}_1, C, \mathsf{FHE.CT}_{M_1})$ to obtain z_2.
- Set $z = (z_1 || z_2)$. Output the i^{th} bit of z, namely, z_i.

Figure 3: Description of program N.

$\underline{\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)}$: On input the secret key $\mathsf{OEE.sk}$, input x and bit b, it executes the following steps.

- It parses $\mathsf{OEE.sk}$ as $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.
- For $\mathsf{ind}_t \in [\lambda]$, it computes the garbled circuit along with the wire keys, $\big(\mathsf{gckt}_{\mathsf{ind}_t}, \{w_{i,0}^{\mathsf{ind}_t}, w_{i,1}^{\mathsf{ind}_t}\}_{i\in[q]}\big) \leftarrow \mathsf{Garble}(1^\lambda, G)$, where $G = G_{(\mathsf{FHE.sk}_b, b)}(\cdot)$ is a circuit that takes as input FHE ciphertexts $(\mathsf{FHE.CT}_0, \mathsf{FHE.CT}_1)$ and outputs a_b, where $a_b \leftarrow \mathsf{FHE.Dec}(\mathsf{FHE.sk}_b, \mathsf{FHE.CT}_b)$. Here, q denotes the total length of two FHE ciphertexts $(\mathsf{FHE.CT}_0, \mathsf{FHE.CT}_1)$.
- For every $i \in [q]$ and $\mathsf{ind}_t \in [\lambda]$, it computes an ABE ciphertext of message pair $(w_{i,0}^{\mathsf{ind}_t}, w_{i,1}^{\mathsf{ind}_t})$ associated to the attribute (x, i, ind_t); $\widetilde{y}_{i,\mathsf{ind}_t} \leftarrow \mathsf{TwoMsgABE.Enc}\big(\mathsf{TwoMsgABE.SK}, (x, i, \mathsf{ind}_t), w_{i,0}^{\mathsf{ind}_t}, w_{i,1}^{\mathsf{ind}_t}\big)$.

Finally, it outputs the encoding $\widetilde{(x,b)} = \big(\mathsf{TwoMsgABE.PP}, \{\mathsf{gckt}\}_{\mathsf{ind}_t\in[\lambda]}, \{\widetilde{y}_{i,\mathsf{ind}_t}\}_{i\in[q],\mathsf{ind}_t\in[\lambda]}\big)$.

$\underline{\mathsf{OEE.Decode}(\widetilde{(M_0, M_1)}, \widetilde{(x,b)})}$: On input the TM encoding $\widetilde{(M_0, M_1)}$ and input encoding $\widetilde{(x,b)}$, it executes the following steps.

- It parses the TM encoding, $\widetilde{(M_0, M_1)} = (\widetilde{N})$ and the input encoding, $\widetilde{(x,b)} = \big(\mathsf{TwoMsgABE.PP}, \{\mathsf{gckt}\}_{\mathsf{ind}_t\in[\lambda]}, \{\widetilde{y}_{i,\mathsf{ind}_t}\}_{i\in[q],\mathsf{ind}_t\in[\lambda]}\big)$.
- For every $\mathsf{ind}_t \in [\lambda]$, it does the following:
  1. For every $i \in [q]$, it executes the decryption procedure of $\mathsf{TwoMsgABE}$ to obtain the wire keys of the garbled circuit, $\widetilde{w}_i^{\mathsf{ind}_t} \leftarrow \mathsf{TwoMsgABE.Dec}(\widetilde{N}, \widetilde{y}_{i,\mathsf{ind}_t})$.
  2. It executes $\mathsf{EvalGC}(\mathsf{gckt}_{\mathsf{ind}_t}, \widetilde{w}_1^{\mathsf{ind}_t}, \ldots, \widetilde{w}_q^{\mathsf{ind}_t})$ to obtain $\mathsf{out}_{\mathsf{ind}_t}$.
  3. If $\mathsf{out}_{\mathsf{ind}_t} \neq \perp$ then output $\mathsf{out} = \mathsf{out}_{\mathsf{ind}_t}$. Otherwise, continue.

This completes the description of the main algorithms. We now describe the auxiliary algorithms.

$\underline{\mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x)}$: The secret key $\mathsf{OEE.sk} = (\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$ punctured at point x is $\mathsf{OEE.sk}_x = (\mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$. That is, the punctured key is same as the original secret key except that the master secret key of $\mathsf{TwoMsgABE}$ is removed. Output $\mathsf{OEE.sk}_x$.

$\underline{\mathsf{OEE.pIEncode}(\mathsf{OEE.sk}_x, x')}$: On input the punctured key $\mathsf{OEE.sk}_x$ and input $x' \neq x$, it executes $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}_x, x', b)$ to obtain the result $\widetilde{(x',b)}$ which is set to be the output.

[Note: The algorithm $\mathsf{OEE.InpEncode}$ *can directly be executed on the punctured key* $\mathsf{OEE.sk}_x$*, input* x *and bit* b *because the master secret key* $\mathsf{TwoMsgABE.SK}$ *is never used during its execution.]*

$\underline{\mathsf{OEE.puncBit}(\mathsf{OEE.sk}, b)}$: On input the secret key $\mathsf{OEE.sk}$ and bit $b \in \{0,1\}$, it first interprets $\mathsf{OEE.sk}$ as $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$. It then outputs the punctured key, $\mathsf{OEE.sk}_b = (\mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_b)$. Output the punctured key $\mathsf{OEE.sk}_b$.

$\underline{\mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_b, x)}$: On input the punctured key $\mathsf{OEE.sk}_b$, it computes $\widetilde{(x,b)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}_b, x, b)$. The result $\widetilde{(x,b)}$ is then output.

[Note: The algorithm $\mathsf{OEE.InpEncode}$ *can directly be executed on the punctured key* $\mathsf{OEE.sk}_b$*, input* x *and bit* b *because the FHE secret key associated to* $\overline{b}$*, namely* $\mathsf{FHE.sk}_{\overline{b}}$*, is never used during the execution.]*

This completes the description of the auxiliary algorithms. We now, in detail, argue that the above scheme satisfies all the properties of an oblivious evaluation encoding scheme. The proof of security is dealt in Appendix D.3.

Correctness. Consider a pair of Turing machines $M_0, M_1 \in \mathcal{M}$, an input $x \in \{0,1\}^*$ and a bit b. Let t^* be the amount of time taken by M_b to execute on x. Suppose $\mathsf{OEE.sk}$ is the output of $\mathsf{OEE.Setup}(1^\lambda)$. Let $\widetilde{N} = \widetilde{N}_{\left(\{\mathsf{FHE.pk}_b, \mathsf{FHE.CT}_{M_b}\}_{b\in\{0,1\}}\right)}$ be the output of $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ and let $\left(\mathsf{TwoMsgABE.PP}, \{\mathsf{gckt}\}_{\mathsf{ind}_t\in[\lambda]}, \{\widetilde{y}_{i,\mathsf{ind}_t}\}_{i\in[q],\mathsf{ind}_t\in[\lambda]}\right)$ be the output of $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$.

- From the correctness of $\mathsf{TwoMsgABE}$ we have the output of $\mathsf{TwoMsgABE.Dec}(\widetilde{N}, \widetilde{y}_{i,\mathsf{ind}_t})$ being the i^{th} wire key of $\mathsf{gckt}_{\mathsf{ind}_t}$ which corresponds to the i^{th} bit of $(\mathsf{FHE.CT}_0, \mathsf{FHE.CT}_1)$. Furthermore, from the correctness of FHE it follows that $\mathsf{FHE.CT}_0$ (resp., $\mathsf{FHE.CT}_1$) is an encryption of $M_0(x)$ (resp., $M_1(x)$), at 2^{ind_t} number of steps, under $\mathsf{FHE.pk}_0$ (resp., $\mathsf{FHE.pk}_1$).
- From the correctness of garbling schemes, it follows that the output of garbled circuit evaluation, $\mathsf{EvalGC}(\mathsf{gckt}_{\mathsf{ind}_t}, \widetilde{w}_1^{\mathsf{ind}_t}, \ldots, \widetilde{w}_q^{\mathsf{ind}_t})$ is $M_b(x)$ when $2^{\mathsf{ind}_t} \geq t^*$ and is $\perp$ otherwise. Since M runs in polynomial time on all inputs, there will exist at least one $\mathsf{ind}_t \in [\lambda]$ such that $2^{\mathsf{ind}_t} \geq t^*$.

Therefore, the output of $\mathsf{OEE.Decode}$ in this case would be $M_b(x)$, as desired.

Efficiency. From the description of the scheme, it follows that $\mathsf{OEE.Setup}(1^\lambda)$ runs in time $\mathrm{poly}(\lambda)$, $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ runs in time $\mathrm{poly}(\lambda, |M_0|, |M_1|)$ and $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$ runs in time $\mathrm{poly}(\lambda, |x|)$. Furthermore, the running time of $\mathsf{OEE.Decode}(\widetilde{(M_0, M_1)}, \widetilde{(x, b)})$ is $\mathrm{poly}(\lambda, t^*)$, where t^* is the time taken to execute M_b on x: the main bottleneck in the running time of $\mathsf{OEE.Decode}$ is the number of the iterations it executes. Further, the i^{th} iteration takes time polynomial in λ and 2^i. If $\mathsf{ind}_t \in [\lambda]$ is the smallest number such that $2^{\mathsf{ind}_t} \geq t^*$ then the number of the iterations in the execution of decode is ind_t. Thus, the total running time of decode is $(\sum_{j=1}^{\mathsf{ind}_t} 2^j)\mathrm{poly}(\lambda) = \mathrm{poly}(\mathrm{t}^*, \lambda)$.

Constant Multiplicative Overhead. Consider a pair of Turing machines $(M_0, M_1) \in \mathcal{M}^2$. The output of $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$, where $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$, is a two-outcome ABE key $\widetilde{N}$ of the program N. From the additive overhead property of $\mathsf{TwoMsgABE}$, the size of $\widetilde{N}$ is $|N| + \mathrm{poly}(\lambda)$. Also by inspection we have, $|N| = |M_0| + |M_1| + \mathrm{poly}(\lambda)$ (which follows from the additive overhead property of FHE). Combining these two facts we get the size of the output encoding of $\mathsf{OEE.TMEncode}$ to be $|M_0| + |M_1| + \mathrm{poly}(\lambda)$.

5 Succinct $\mathsf{i}\mathcal{O}$ with Constant Multiplicative Overhead

Let $\mathsf{OEE} = (\mathsf{OEE.Setup}, \mathsf{OEE.InpEncode}, \mathsf{OEE.TMEncode}, \mathsf{OEE.Decode})$ be an OEE scheme with additive overhead. Let $\mathsf{i}\mathcal{O}$ be an indistinguishability obfuscator for general circuits. Let PRF be a puncturable PRF family. Using these primitives, we now give a construction of a succinct indistinguishability obfuscator with additive overhead. We denote it by SuccIO.

Construction. Let $\mathcal{M}$ denote the family of turing machines. On input the security parameter and a turing machine $M \in \mathcal{M}$, $\mathsf{SuccIO}(1^\lambda, M)$ computes the following:

- $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda, T)$, where T is the bound on the running time of M.
- $\widetilde{(M, M)} \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M, M)$.
- $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C_{[K,\mathsf{OEE.sk}]}\right)$, where K is a randomly chosen key for the puncturable PRF family and $C_{[K,\mathsf{OEE.sk}]}$ is the circuit described in Figure 4.

The output of the obfuscator is $\left(\widetilde{(M, M)}, \widetilde{C}\right)$.

To evaluate the obfuscated machine on an input x, the evaluator first computes $\widetilde{C}(x)$ to obtain $\widetilde{(x, 0)}$. Next, it computes $y \leftarrow \mathsf{OEE.Decode}\left(\widetilde{(M, M)}, \widetilde{(x, 0)}\right)$ and outputs y.

```
                    $C_{[K,\mathsf{OEE.sk}]}(x)$
1. Compute $r \leftarrow \mathsf{PRF}_K(x\|0)$.
2. Compute $\widetilde{(x,0)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 0)$ using randomness $r$.
3. Output $\widetilde{(x,0)}$.
```

Figure 4: Circuit $C_{[K,\mathsf{OEE.sk}]}$.

Theorem 4. *If* PRF *is a sub-exponentially secure puncturable PRF,* OEE *is a sub-exponentially secure OEE scheme with additive overhead and* $i\mathcal{O}$ *is a sub-exponentially secure indistinguishability obfuscator for general circuits, then* SuccIO *is a succinct indistinguishability obfuscator with additive overhead.*

Below we argue the efficiency of our construction. The proof of correctness and security is deferred to Appendix E.

Efficiency. The size of the obfuscated program is $|\widetilde{(M,M)}| + |\widetilde{C}|$. From the efficiency of the machine encoding algorithm of the OEE scheme, it follows that $|\widetilde{(M,M)}| = c \cdot |M| + \text{poly}(\lambda)$ for some constant c. Further, from the efficiency of the $i\mathcal{O}$ scheme, it follows that $|\widetilde{C}| = \text{poly}(\lambda, |C_{[K,\mathsf{OEE.sk}]}|)$. Further, from the efficiency of the input encoding algorithm of the OEE scheme, it follows that $|C_{[K,\mathsf{OEE.sk}]}| = \text{poly}(\lambda, L)$, where L is the bound on the input length.

Putting it all together, we have that the size of the obfuscated program is $c \cdot |M| + \text{poly}(\lambda, L)$, as required.

References

[1] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.

[2] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *ACM CCS*, 2014.

[3] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.

[4] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *TCC*, 2016A.

[5] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, 2015.

[6] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

[7] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[8] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM*, 2012.

[9] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

[10] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.

[11] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[12] Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors. *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. ACM, 2013.

[13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

[14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, 2014.

[15] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.

[16] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[17] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. *IACR Cryptology ePrint Archive*, 2015:388, 2015.

[18] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. In *STOC*, 2015.

[19] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 519–535, 2013.

[20] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Computation-trace indistinguishability obfuscation and its applications. *IACR Cryptology ePrint Archive*, 2015:406, 2015.

[21] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6), 1985.

[22] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[23] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *CRYPTO*, 2014.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[25] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[26] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[27] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Boneh et al. [12], pages 555–564.

[28] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, 2013.

[29] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Boneh et al. [12], pages 545–554.

[30] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *ACM STOC*, 2008.

[31] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *TCC*, 2015.

[32] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.

[33] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, 2015.

[34] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. *IACR Cryptology ePrint Archive*, 2015:720, 2015.

[35] Yehuda Lindell and Benny Pinkas. A proof of security of yao?s protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[36] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, 1990.

[37] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, 2014.

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

[39] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[40] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *ACM STOC*, 2014.

[41] Amit Sahai and Mark Zhandry. Obfuscating low-rank matrix branching programs. Technical report, Cryptology ePrint Archive, Report 2014/773, 2014. http://eprint. iacr. org, 2014.

[42] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014.

[43] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[44] Joe Zimmerman. How to obfuscate programs directly. In *EUROCRYPT*, 2015.

A Preliminaries (cont'd)

A.1 Turing machines (TMs)

A Turing machine is a 7-tuple $M = \langle Q, \Sigma_{\text{inp}}, \Sigma_{\text{tape}}, \perp, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ where Q and Σ_{tape} are finite sets with the following properties:

1. Q is the set of finite states.
2. Σ_{inp} is the set of input symbols.
3. Σ_{tape} is the set of tape symbols.
4. $\perp$ denotes the blank symbol.
5. $\delta : Q \times \Sigma_{\text{tape}} \to Q \times \Sigma_{\text{tape}} \times \{+1, -1\}$ is the transition function.
6. $q_0 \in Q$ is the start state.
7. $q_{\text{acc}} \in Q$ is the accept state.
8. $q_{\text{rej}} \in Q$ is the reject state, where $q_{\text{acc}} \neq q_{\text{rej}}$.

Turing machines to circuits. A Turing machine running in time at most $T(n)$ on inputs of size n, can be transformed into a circuit of input length n and of size $O\big((T(n))^2\big)$. This theorem proved by Pippenger and Fischer [38] is stated below.

Theorem 5. *Any Turing machine M running in time at most $T(n)$ for all inputs of size n, can be transformed into a circuit $C_M : \{0,1\}^n \to \{0,1\}$ such that (1) $C_M(x) = M(x)$ for all $x \in \{0,1\}^n$, and (2) the size of C_M is $|C_M| = O\big((T(n))^2\big)$. We denote this transformation procedure as* TMtoCKT.

We use the notation $\mathsf{RunTime}$ to denote the running time of a TM on a specific input. More formally, $\mathsf{RunTime}(M, x)$ outputs the time taken by M to run on x.

In this work, we only consider TMs which run in polynomial time on all its inputs, i.e., there exists a polynomial p such that the running time is at most $p(n)$ for every input of length n. We note that our schemes can be generalized in a natural way to the Turing machines for which this restriction does not apply.

A.2 Puncturable Pseudorandom Functions

A pseudorandom function family F consisting of functions of the form $\mathsf{PRF}_K(\cdot)$, that is defined over input space $\{0,1\}^{\eta(\lambda)}$, output space $\{0,1\}^{\chi(\lambda)}$ and key K in the key space $\mathcal{K}$, is said to be a *secure puncturable PRF family* if there exists a PPT algorithm $\mathsf{PRFPunc}$ that satisfies the following properties:

- **Functionality preserved under puncturing.** $\mathsf{PRFPunc}$ takes as input a PRF key K, sampled from $\mathcal{K}$, and an input $x \in \{0,1\}^{\eta(\lambda)}$ and outputs K_x such that for all $x' \neq x$, $\mathsf{PRF}_{K_x}(x') = \mathsf{PRF}_K(x')$.
- **Pseudorandom at punctured points.** For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(1^\lambda)$ outputs an input $x \in \{0,1\}^{\eta(\lambda)}$, consider an experiment where $K \xleftarrow{\$} \mathcal{K}$ and $K_x \leftarrow \mathsf{PRFPunc}(K, x)$. Then for all sufficiently large $\lambda \in \mathbb{N}$, for a negligible function μ,
$$\left|\Pr[\mathcal{A}_2(K_x, x, \mathsf{PRF}_K(x)) = 1] - Pr[\mathcal{A}_2(K_x, x, U_{\chi(\lambda)}) = 1]\right| \leq \mu(\lambda)$$
where $U_{\chi(\lambda)}$ is a string drawn uniformly at random from $\{0,1\}^{\chi(\lambda)}$.

As observed by [13, 15, 32], the GGM construction [26] of PRFs from one-way functions yields puncturable PRFs.

Theorem 6 ([26, 13, 15, 32])**.** *If μ-secure one-way functions*[12] *exist, then for all polynomials $\eta(\lambda)$ and $\chi(\lambda)$, there exists a μ-secure puncturable PRF family that maps $\eta(\lambda)$ bits to $\chi(\lambda)$ bits.*

A.3 Garbling schemes

Yao in his seminal work [43, 35] proposed the notion of garbled circuits as a solution to the problem of secure two party computation. Recently, Bellare et al. [8] formalized this by calling them garbling schemes. We define garbling schemes next. The syntax of our scheme is similar to the definition of Bellare et al.

A garbling scheme GC for a class of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n \in \mathbb{N}}$ consists of two PPT algorithms namely $(\mathsf{Garble}, \mathsf{EvalGC})$.

- **Garbling algorithm**, $\mathsf{Garble}(1^\lambda, C \in \mathcal{C})$: On input a security parameter λ in unary and a circuit $C \in \mathcal{C}_n$ of input length n, it outputs a garbled circuit along with its wire keys, $(\mathsf{gckt}, \{w_{i,0}, w_{i,1}\}_{i\in[n]})$.
- **Garbled circuit evaluation algorithm**, $\mathsf{EvalGC}(\mathsf{gckt}, \{w_{i,x_i}\}_{i\in[n]})$: On input the garbled circuit gckt along with the input wire keys $\{w_{i,x_i}\}_{i\in[n]}$ corresponding to an input x, it outputs out.

The correctness property of a garbling scheme dictates that for every $C \in \mathcal{C}_n$, the output of the evaluation procedure $\mathsf{EvalGC}(\mathsf{gckt}, \{w_{i,x_i}\}_{i\in[n]})$ is $C(x)$, where $(\mathsf{gckt}, \{w_{i,0}, w_{i,1}\}_{i\in[n]}) \leftarrow \mathsf{Garble}(1^\lambda, C)$.

[12] We say that a one-way function family is μ-secure if the probability of inverting a one-way function, that is sampled from the family, is at most $\mu(\lambda)$.

Security. A garbling scheme is said to be secure if the joint distribution of garbled circuit along with the wire keys, corresponding to some input, reveals only the output of the circuit and nothing else. This can be formalized in the form of a simulation-based notion as given below.

Definition 9. *A garbling scheme* $\mathsf{GC} = (\mathsf{Garble}, \mathsf{EvalGC})$ *for a class of circuits* $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *is said to be secure if there exists a simulator* $\mathsf{SimGarble}$ *such that for any* $C \in \mathcal{C}_n$ *the following two distributions are computationally distinguishable.*

- $\{\mathsf{SimGarble}(1^\lambda, 1^{|C|}, C(x))\}$
- $\{(\mathsf{gckt}, \{w_{i,x_i}\}_{i\in[n]})\}$, *where* $(\mathsf{gckt}, \{w_{i,0}, w_{i,1}\}_{i\in[n]}) \leftarrow \mathsf{Garble}(1^\lambda, C)$.

A.4 Fully Homomorphic Encryption for circuits

Another main tool that we use in one of our constructions is a fully homomorphic encryption scheme (FHE). First proposed more than three decades ago, a construction of this primitive was conceived in 2009 by Gentry [24].

A public key fully homomorphic encryption (FHE) scheme for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ and message space $\mathsf{MSG} = \{\mathsf{MSG}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of four PPT algorithms, namely, $(\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$. The syntax of the algorithms are described below.

- **Setup,** $\mathsf{FHE.Setup}(1^\lambda)$: On input a security parameter 1^λ it outputs a public key-secret key pair $(\mathsf{FHE.pk}, \mathsf{FHE.sk})$.
- **Encryption,** $\mathsf{FHE.Enc}(\mathsf{FHE.pk}, m \in \mathsf{MSG}_\lambda)$: On input public key $\mathsf{FHE.pk}$ and message $m \in \mathsf{MSG}_\lambda$, it outputs a ciphertext denoted by $\mathsf{FHE.CT}$.
- **Evaluation,** $\mathsf{FHE.Eval}(\mathsf{FHE.pk}, C \in \mathcal{C}, \mathsf{FHE.CT})$: On input public key $\mathsf{FHE.pk}$, a circuit $C \in \mathcal{C}_\lambda$ and a FHE ciphertext $\mathsf{FHE.CT}$, it outputs the evaluated ciphertext $\widetilde{\mathsf{FHE.CT}}$.
- **Decryption,** $\mathsf{FHE.Dec}(\mathsf{FHE.sk}, \mathsf{FHE.CT})$: On input the secret key $\mathsf{FHE.sk}$ and a ciphertext $\mathsf{FHE.CT}$, it outputs the decrypted value out.

The correctness property guarantees the following, where $(\mathsf{FHE.pk}, \mathsf{FHE.sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$ and $\mathsf{FHE.CT} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}, m \in \mathsf{MSG}_\lambda)$:

- $m \leftarrow \mathsf{FHE.Dec}(\mathsf{FHE.sk}, \mathsf{FHE.CT})$
- For any circuit $C \in \mathcal{C}_\lambda$ where the input length of C is $|m|$, we have $C(m) \leftarrow \mathsf{FHE.Dec}(\mathsf{FHE.sk}, \mathsf{FHE.Eval}(\mathsf{FHE.pk}, C, \mathsf{FHE.CT}))$.

The security notion of an FHE scheme is identical to the definition of semantic security of a public key encryption scheme.

An FHE scheme should also satisfy the so called compactness property. At a high level, the compactness property ensures that the length of the ciphertext output by $\mathsf{FHE.Eval}(\mathsf{FHE.pk}, C, \cdot)$ is independent of the size of C.

FHE with Additive Overhead. A fully homomorphic encryption scheme is said to satisfy additive overhead property if the size of ciphertext of a message m is $|m| + \mathrm{poly}(\lambda)$. An FHE scheme with additive overhead can achieved generically starting from any FHE scheme. Suppose $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ be any FHE scheme. Then we can define a FHE scheme $\mathsf{FHE}_{\mathsf{a.o.}} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}^*, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ as follows. The encryption algorithm $\mathsf{FHE.Enc}^*\Big(\mathsf{FHE.pk}, m\Big)$ outputs $(\mathsf{Sym.Enc}(\mathsf{Sym.sk}, m), \mathsf{FHE.Enc}(\mathsf{FHE.pk}, \mathsf{Sym.sk}))$, where (i) $\mathsf{FHE.pk}$ is a public key produced by $\mathsf{FHE.Setup}$, (ii) Sym is a (symmetric) encryption algorithm with $\mathsf{Sym.sk}$ being the (symmetric) key. Observe that $\mathsf{FHE}_{\mathsf{a.o.}}$ satisfies additive overhead property if we use a symmetric encryption scheme that satisfies additive overhead property.

B Formal Definition of 1-Key ABE for TMs

A 1-key ABE for Turing machines scheme, defined for a class of Turing machines $\mathcal{M}$, consists of four PPT algorithms, $\mathsf{1ABE} = (\mathsf{1ABE.Setup}, \mathsf{1ABE.KeyGen}, \mathsf{1ABE.Enc}, \mathsf{1ABE.Dec})$. We denote the associated message space to be MSG. The syntax of the algorithms is given below.

1. **Setup,** $\mathsf{1ABE.Setup}(1^\lambda)$: On input a security parameter λ in unary, it outputs a public key-secret key pair $(\mathsf{1ABE.PP}, \mathsf{1ABE.SK})$.
2. **Key Generation,** $\mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M \in \mathcal{M})$: On input a secret key $\mathsf{1ABE.SK}$ and a TM $M \in \mathcal{M}$, it outputs an ABE key $\mathsf{1ABE}.sk_M$.
3. **Encryption,** $\mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg})$: On input the public parameters $\mathsf{1ABE.PP}$, attribute $x \in \{0,1\}^*$ and message $\mathsf{msg} \in \mathsf{MSG}$, it outputs the ciphertext $\mathsf{1ABE.CT}_{(x,\mathsf{msg})}$.
4. **Decryption,** $\mathsf{1ABE.Dec}(\mathsf{1ABE}.sk_M, \mathsf{1ABE.CT}_{(x,\mathsf{msg})})$: On input the ABE key $\mathsf{1ABE}.sk_M$ and encryption $\mathsf{1ABE.CT}_{(x,\mathsf{msg})}$, it outputs the decrypted result out.

Correctness. The correctness property dictates that the decryption of a ciphertext of (x, msg) using an ABE key of M yields the message msg if $M(x) = 1$. In formal terms, the output of the decryption procedure $\mathsf{1ABE.Dec}(\mathsf{1ABE}.sk_M, \mathsf{1ABE.CT}_{(x,\mathsf{msg})})$ is (always) msg if $M(x) = 1$, where $(\mathsf{1ABE.SK}, \mathsf{1ABE.PP}) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$, $\mathsf{1ABE}.sk_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M \in \mathcal{M})$ and $\mathsf{1ABE.CT}_{(x,\mathsf{msg})} \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg})$.

Security. The security framework we consider is identical to the indistinguishability based security notion of ABE for circuits except that (i) the key queries correspond to Turing machines instead of circuits and (ii) the adversary is only allowed to make a single key query. Furthermore, we only consider the setting when the adversary submits both the challenge message pair as well as the key query at the beginning of the game itself. We term this *weak selective security.* We formally define this below.

The security is defined in terms of the following security experiment between a challenger and a PPT adversary. We denote the challenger by Ch and the adversary by $\mathcal{A}$.

$\underline{\mathsf{Expt}^{\mathsf{1ABE}}_{\mathcal{A}}(1^\lambda, b \in \{0,1\})}$:

1. $\mathcal{A}$ sends to Ch a tuple consisting of a Turing machine M, an attribute x and two messages $(\mathsf{msg}_0, \mathsf{msg}_1)$.
2. The challenger Ch replies to $\mathcal{A}$ with the public key $\mathsf{1ABE.PP}$, an ABE key $\mathsf{1ABE}.sk_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M)$ for machine M and the challenge ciphertext $\mathsf{1ABE.CT}_{(x,\mathsf{msg}_b)} \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg}_b)$, where $(\mathsf{1ABE.SK}, \mathsf{1ABE.PP}) \leftarrow \mathsf{1ABE}, \mathsf{Setup}(1^\lambda)$.
3. The experiment terminates when the adversary outputs the bit b'.

We say that a 1-key ABE for TMs scheme is said to be weak-selectively secure if any PPT adversary can guess the challenge bit only with negligible probability.

Definition 10. *A 1-key attribute based encryption for TMs scheme is said to be* **weak-selectively secure** *if* $|\Pr[0 \leftarrow \mathsf{Expt}^{\mathsf{1ABE}}_{\mathcal{A}}(1^\lambda, 0)] - \Pr[0 \leftarrow \mathsf{Expt}^{\mathsf{1ABE}}_{\mathcal{A}}(1^\lambda, 1)]| \leq \mathsf{negl}(\lambda)$*, where* negl *is a negligible function.*

Remark 4. *Henceforth, we will not explicitly use the term "weak-selective" when referring to the security of ABE schemes.*

C Proof of Security of 1-Key ABE for TMs

Consider the following sequence of hybrids. The first hybrid corresponds to the real experiment (as described in the security game) when the challenger picks a bit b at random and sets the challenge bit to be b. We then describe a series of intermediate hybrids such that every two consecutive hybrids are computationally indistinguishable. In the final hybrid, the challenger picks a bit b at random but sets the challenge bit to be 0. At this point the probability that the PPT adversary $\mathcal{A}$ can guess the bit b is $\frac{1}{2}$.

We denote $\mathsf{adv}_{\mathcal{A},i}$ to be the advantage of $\mathcal{A}$ in Hyb_i.

Hybrid Hyb_1: The challenger receives from $\mathcal{A}$, a Turing machine M, an attribute x and two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathsf{MSG}$. The challenger then responds with the public key $\mathsf{1ABE.PP}$, an ABE key of M, namely $\mathsf{1ABE}.sk_M$ and an encryption of msg_b w.r.t attribute x, namely $\mathsf{1ABE.CT}^*$, where b is picked at random. All the parameters are generated honestly by the challenger.

The output of the hybrid is the output of the adversary.

Hybrid Hyb_2: The verification key $\mathsf{VK}_{\mathsf{tm}}$ is replaced by a verification key that only verifies on the root of the accumulator storage, initialized with the TM M, and rejects signatures on all other messages. The rest of the hybrid is the same as the previous hybrid.

The challenger upon receiving a TM M, attribute x and messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathsf{MSG}$, does the following. It first picks a bit b at random. It generates the accumulator and the iterator parameters $\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0$ as in the setup algorithm. It then initializes the accumulator storage with the Turing machine M as follows: as before, let $\ell_{\mathsf{tm}} = |M|$ be the length of the Turing machine. It computes $\widetilde{store}_j = \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1, M_j)$, $aux_j = \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1)$, $\widetilde{w}_j = \mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_{j-1}, \mathsf{inp}_j, j-1, aux_j)$ for $1 \leq j \leq \ell_{\mathsf{tm}}$. Finally, it sets $\mathbf{w} = \widetilde{w}_{\ell_{\mathsf{tm}}}$.

It then executes the setup of splittable signatures scheme, $(\mathsf{SK}_{\mathsf{tm}}, \mathsf{VK}_{\mathsf{tm}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$. It then executes the split algorithm of the signatures scheme to obtain, $(\sigma_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{SK}_{\backslash \mathbf{y}}, \mathsf{VK}_{\backslash \mathbf{y}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}_{\mathsf{tm}}, \mathbf{y} = (v_0, q_0, \mathbf{w}, 0))$. Of particular interest to us is $\sigma_{\mathsf{tm}}^{\mathbf{y}}$, which is the (deterministic) signature on $\mathbf{y}$ and $\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}$, which is the verification key that only validates the message-signature pair $(\mathbf{y}, \sigma_{\mathsf{tm}}^{\mathbf{y}})$ and invalidates all other message-signature pairs. It finally sets the public key as $\big(\mathsf{1ABE.PP} = (\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)\big)$.

The challenger then sets $\mathsf{1ABE}.sk_M = (M, \mathbf{w}, \sigma_{\mathsf{tm}}^{\mathbf{y}}, v_0)$. It generates the challenge ciphertext by computing $\mathsf{1ABE.CT}^* \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg}_b)$. It then sends $(\mathsf{1ABE.PP}, \mathsf{1ABE}.sk_M, \mathsf{1ABE.CT}^*)$ to $\mathcal{A}$.

Claim 1. *Assuming that* $\mathsf{SplScheme}$ *satisfies* $\mathsf{VK}_{\mathrm{one}}$ *indistinguishability (Definition 4), for any PPT adversary* $\mathcal{A}$ *we have* $|\mathsf{adv}_{\mathcal{A},1} - \mathsf{adv}_{\mathcal{A},2}| \leq \mathsf{negl}(\lambda)$.

Proof. The only message-signature pair, with respect to the instantiation of the key pair $(\mathsf{SK}_{\mathsf{tm}}, \mathsf{VK}_{\mathsf{tm}})$, provided to the adversary $\mathcal{A}$ is $(\mathbf{y}, \sigma_{\mathsf{tm}}^{\mathbf{y}})$. Even with this additional information, the verification keys $\mathsf{VK}_{\mathsf{tm}}$ from $\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}$, defined as in Hyb_1 and Hyb_2, are computationally indistinguishable from the $\mathsf{VK}_{\mathrm{one}}$ property of $\mathsf{SplScheme}$. The proof of the claim follows. □

Hybrid Hyb_3: The program $\mathsf{SignProg}$, which is part of the encryption process, is now modified with the output hardwired into it. The rest of the hybrid is as before.

The challenger upon receiving a TM M, attribute x and messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathsf{MSG}$, does the following. It first picks a bit b at random. It sets $\mathsf{msg}^* = \mathsf{msg}_b$. It then computes $\mathsf{1ABE.PP} = (\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)$ as in Hyb_2. Further, it computes the ABE key of M, namely $\mathsf{1ABE}.sk_M = (M, \mathbf{w}, \sigma_{\mathsf{tm}}^{\mathbf{y}}, v_0)$, as in Hyb_2.

It then samples a PRF key K_A at random. It computes the obfuscation of the program $U_x(\cdot)$, $N \leftarrow i\mathcal{O}(\mathsf{NxtMsg}\{U_x(\cdot), \mathsf{msg}^*, \mathsf{PP}_{\mathsf{Acc}}, \mathsf{PP}_{\mathsf{Itr}}, K_A\})$ where $U_x(\cdot)$ is defined as in $\mathsf{1ABE.Enc}$ and NxtMsg is defined in Figure 1. From here onwards, the challenger deviates from the honest execution of the encryption algorithm. It generates the signing key-verification key pair $(\mathsf{SK}_0, \mathsf{VK}_0) \leftarrow \mathsf{SetupSpl}(1^\lambda; r_A)$, where r_A is the output of $F(K, 0)$. It computes the signature $\sigma_0 \leftarrow \mathsf{SignSpl}(\mathsf{SK}_0, \mathbf{y} = (v_0, q_0, \mathbf{w}, 0))$. As before, it generates $(\sigma_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{SK}_{\backslash \mathbf{y}}, \mathsf{VK}_{\backslash \mathbf{y}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}_{\mathsf{tm}}, \mathbf{y} = (v_0, q_0, \mathbf{w}, 0))$. It then computes the obfuscation of the program $S^* \leftarrow (\mathsf{HybSgn}\{\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \sigma_0\})$ where HybSgn is defined in Figure 5. It sets the ciphertext $\mathsf{1ABE.CT}^* = (N, S^*)$. The challenger then sends $(\mathsf{1ABE.PP}, \mathsf{1ABE}.sk_M, \mathsf{1ABE.CT}^*)$ to $\mathcal{A}$.

Claim 2. *Assuming the security of the scheme* $i\mathcal{O}$, *for any PPT adversary* $\mathcal{A}$ *we have that* $|\mathsf{adv}_{\mathcal{A},2} - \mathsf{adv}_{\mathcal{A},3}| \leq \mathsf{negl}(\lambda)$.

Proof. Suppose $S \leftarrow iO(\mathsf{SignProg}\{K_A, \mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}\})$ as in Hyb_2 and $S^* \leftarrow iO(\mathsf{HybSgn}\{\mathsf{VK}^{\mathbf{w}}_{\mathsf{tm}}, \sigma_0\})$ as in Hyb_3. To prove the claim, it suffices to show that it is computationally hard to distinguish S and S^*. This further reduces, courtesy security of iO, to showing that $\mathsf{SignProg}\{K_A, \mathsf{VK}_{\mathsf{tm}}\}$ and $\mathsf{HybSgn}\{\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \sigma_0\}$ are functionally equivalent. Consider the input (y, σ) to both the programs. There are two cases to consider:

- **Case** $(y, \sigma) \neq (\mathbf{y}, \sigma^{\mathbf{y}}_{\mathsf{tm}})$: The program $\mathsf{SignProg}\{K_A, \mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}\}(y, \sigma)$ outputs $\perp$ because (y, σ) is invalid with respect to $\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}$. For the same reason, program $\mathsf{HybSgn}\{\mathsf{VK}^{\mathbf{w}}_{\mathsf{tm}}, \sigma_0\}(y, \sigma)$ also outputs $\perp$.
- **Case** $(y, \sigma) = (\mathbf{y}, \sigma^{\mathbf{y}}_{\mathsf{tm}})$: The program $\mathsf{SignProg}\{K_A, \mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}\}(y, \sigma)$ outputs the signature σ_0 computed by first running $r_A \leftarrow F(K, 0)$, then $(\mathsf{SK}_0, \mathsf{VK}_0) \leftarrow \mathsf{SetupSpl}(1^\lambda)$ and finally $\sigma_0 \leftarrow \mathsf{SignSpl}(\mathsf{SK}_0, \mathbf{y})$. The program HybSgn outputs the hardwired σ_0, where σ_0 is pre-computed exactly as in $\mathsf{SignProg}$.

Thus the programs $\mathsf{SignProg}$ and HybSgn are functionally equivalent. This proves the claim. $\square$

$\mathsf{HybSgn}\{\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \sigma_0\}$

Constants: PRF key K_A, verification key $\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}$ and signature σ_0.
Input: Message y and a signature σ_{tm}.

1. If $\mathsf{VerSpl}(\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, y, \sigma_{\mathsf{tm}}) = 0$ then output $\perp$. Otherwise output σ_0.

Figure 5: Program HybSgn

Hybrid Hyb_4: This is identical to Hyb_3 except that the message msg^* to be encrypted is now set to msg_0, where $(\mathsf{msg}_0, \mathsf{msg}_1)$ is the challenge message pair submitted by the adversary. Recall that in Hyb_3, msg^* was set to msg_b, where b is picked at random.

Claim 3. *From Theorem 2, we have* $|\mathsf{adv}_{\mathcal{A},3} - \mathsf{adv}_{\mathcal{A},4}| \leq \mathsf{negl}(\lambda)$

Proof. Assume that the claim is not true. We then construct a reduction $\mathcal{B}$ that uses the adversary $\mathcal{A}$ to contradict Theorem 2.

$\mathcal{A}$ first sends the Turing machine $M \in \mathcal{M}$, input x and message pair $(\mathsf{msg}_0, \mathsf{msg}_1) \in \mathsf{MSG}^2$ to $\mathcal{B}$. The reduction then obtains a sample from the distribution $\mathcal{D}_{M,x,\mathsf{msg}_b}$, where b is either picked at random or set to 0. It then parses the sample as $\big(N, \mathbf{w}, v_0, \sigma_0, store_0, \mathsf{init} = (\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}})\big)$. The reduction $\mathcal{B}$ then samples a signature key-verification key pair by running the setup of $\mathsf{SplScheme}$, $(\mathsf{SK}_{\mathsf{tm}}, \mathsf{VK}_{\mathsf{tm}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$. It then executes the split algorithm, $(\sigma^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{SK}_{\backslash \mathbf{y}}, \mathsf{VK}_{\backslash \mathbf{y}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}_{\mathsf{tm}}, \mathbf{y} = (v_0, q_0, \mathbf{w}, 0)$. Finally, $\mathcal{B}$ generates the obfuscation of the program HybSgn described in Figure 5, $S^* \leftarrow (\mathsf{HybSgn}\{\mathsf{VK}^{\mathbf{w}}_{\mathsf{tm}}, \sigma_0\})$. The reduction then prepares the ABE public key, attribute key and challenge ciphertext as below:

- The public key is set to be $\mathsf{1ABE.PP} = (\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)$.
- The attribute key of M to be $\mathsf{1ABE}.sk_M = (M, \mathbf{w}, \widetilde{store}_0, \sigma^{\mathbf{y}}_{\mathsf{tm}}, v_0)$.
- The challenge ciphertext is set to be $\mathsf{1ABE.CT}^* = (N, S^*)$.

$\mathcal{B}$ then sends $(\mathsf{1ABE.PP}, \mathsf{1ABE}.sk_M, \mathsf{1ABE.CT}^*)$ across to $\mathcal{A}$. The output of $\mathcal{B}$ is set to be the output of $\mathcal{A}$.

If the bit b in $\mathcal{D}_{M,x,\mathsf{msg}_b}$ is picked at random then we are in Hyb_3 and if it is set to be 0 then we are in Hyb_4. From our hypothesis (that the claim is not true), this means that the hybrids Hyb_3 and Hyb_4 are computationally indistinguishable. Thus we arrive at a contradiction of Theorem 2. This completes the proof. $\square$

The probability that $\mathcal{A}$ outputs the bit b in Hyb_4 is $1/2$. From Claims 1, 2 and 3, we have that the probability that $\mathcal{A}$ outputs bit b in Hyb_1 is negligibly close to $1/2$. This completes the proof.

D 1-Key Two-Outcome ABE for TMs

D.1 Definition

A 1-key two-outcome ABE for TMs scheme, defined for a class of Turing machines $\mathcal{M}$ and message space MSG, consists of four PPT algorithms, $(\mathsf{TwoMsgABE.Setup}, \mathsf{TwoMsgABE.KeyGen}, \mathsf{TwoMsgABE.Enc}, \mathsf{TwoMsgABE.Dec})$. The syntax of the algorithms is given below.

1. **Setup,** $\mathsf{TwoMsgABE.Setup}(1^\lambda)$: On input a security parameter λ in unary, it outputs a secret key $\mathsf{TwoMsgABE.SK}$ and public key $\mathsf{TwoMsgABE.PP}$.
2. **Key Generation,** $\mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, M \in \mathcal{M})$: On input a secret key $\mathsf{TwoMsgABE.SK}$ and a TM $M \in \mathcal{M}$, it outputs an ABE key $\mathsf{TwoMsgABE}.sk_M$.
3. **Encryption,** $\mathsf{TwoMsgABE.Enc}(\mathsf{TwoMsgABE.PP}, x, \mathsf{msg}_0, \mathsf{msg}_1)$: On input the public key $\mathsf{TwoMsgABE.PP}$, attribute $x \in \{0,1\}^*$ and a pair of messages $(\mathsf{msg}_0 \in \mathsf{MSG}, \mathsf{msg}_1 \in \mathsf{MSG})$, it outputs the ciphertext $\mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)}$.
4. **Decryption,** $\mathsf{TwoMsgABE.Dec}(\mathsf{TwoMsgABE}.sk_M, \mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)})$: On input the ABE key $\mathsf{1ABE}.sk_M$ and ciphertext $\mathsf{1ABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)}$, it outputs the decrypted value out.

Correctness. The correctness property dictates that the decryption of a ciphertext of $(x, \mathsf{msg}_0, \mathsf{msg}_1)$, using the ABE key of M, yields the message msg_0 if $M(x) = 0$, otherwise it outputs msg_1. Formally, $\mathsf{TwoMsgABE.Dec}(\mathsf{TwoMsgABE}.sk_M, \mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)})$ is (always) msg_0 if $M(x) = 0$ or msg_1 if $M(x) = 1$, where

- $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}) \leftarrow \mathsf{TwoMsgABE.Setup}(1^\lambda)$,
- $\mathsf{TwoMsgABE}.sk_M \leftarrow \mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, M \in \mathcal{M})$ and
- $\mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)} \leftarrow \mathsf{TwoMsgABE.Enc}(\mathsf{TwoMsgABE.PP}, x, \mathsf{msg}_0, \mathsf{msg}_1)$.

Security. As in the single-message case, we define an indistinguishability based security notion of 1-key two-outcome ABE scheme. The security notion is formalized in the form of the following security experiment between a challenger and a PPT adversary. We denote the challenger by Ch and the adversary by $\mathcal{A}$.

$\underline{\mathsf{Expt}_{\mathcal{A}}^{\mathsf{TwoMsgABE}}(1^\lambda, b \in \{0,1\})}$:

1. $\mathcal{A}$ sends to Ch a key query M, and input comprising of the attribute x and two pairs of messages $\Big((\mathsf{msg}_{0,0}, \mathsf{msg}_{0,1}), (\mathsf{msg}_{1,0}, \mathsf{msg}_{1,1})\Big)$.
2. Ch checks if (i) $M(x) = 0$ and $\mathsf{msg}_{0,0} = \mathsf{msg}_{1,0}$ or if (ii) $M(x) = 1$ and $\mathsf{msg}_{0,1} = \mathsf{msg}_{1,1}$. If both the conditions are not satisfied then Ch aborts the experiment. Otherwise, it replies to $\mathcal{A}$ with the public key $\mathsf{TwoMsgABE.PP}$, two-outcome ABE predicate key $(\mathsf{TwoMsgABE}.sk_M) \leftarrow \mathsf{1ABE.KeyGen}(1^\lambda, M)$ and the challenge ciphertext $\mathsf{TwoMsgABE.CT}_{(x,\mathsf{msg}_{b,0},\mathsf{msg}_{b,1})} \leftarrow \mathsf{TwoMsgABE.Enc}(\mathsf{TwoMsgABE.PP}, x, \mathsf{msg}_{b,0}, \mathsf{msg}_{b,1})$.
3. The experiment terminates when the adversary outputs the bit b'.

We are now ready to define the security of 1-key two-outcome ABE for TMs scheme. We say that a 1-key two-outcome ABE for TMs scheme is said to be secure if any PPT adversary can guess the challenge bit only with negligible probability.

Definition 11. *A 1-key two-outcome ABE for TMs scheme is said to be secure if* $|\mathsf{Pr}[0 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{TwoMsgABE}}(1^\lambda, 0)] - \mathsf{Pr}[0 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{TwoMsgABE}}(1^\lambda, 1)]| \leq \mathsf{negl}(\lambda)$*, where* negl *is a negligible function.*

1-Key Two-Outcome ABE for TMs with Constant Multiplicative Overhead. We define the notion of 1-Key Two-Outcome ABE for TMs with constant multiplicative overhead. The size of the attribute key of M is $c \cdot |M| + \mathrm{poly}(\lambda)$ for a constant c. The formal definition is provided below.

Definition 12 (Constant Multiplicative Overhead)**.** *A 1-key two-outcome ABE for TMs scheme,* $\mathsf{TwoMsgABE}$*, defined for a class of Turing machines* $\mathcal{M}$*, is said to have constant multiplicative overhead if* $|\mathsf{TwoMsgABE}.sk_M| = c \cdot |M| + \mathrm{poly}(\lambda)$*, for a constant* c*, where* $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}) \leftarrow \mathsf{TwoMsgABE.Setup}(1^\lambda)$ *and* $\mathsf{TwoMsgABE}.sk_M \leftarrow \mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, M \in \mathcal{M})$*.*

If the constant overhead in the TM size is 1 then we say that 1-key two-outcome ABE for TMs satisfies additive overhead property.

Definition 13 (Additive Multiplicative Overhead)**.** *A 1-key two-outcome ABE for TMs scheme,* $\mathsf{TwoMsgABE}$*, defined for a class of Turing machines* $\mathcal{M}$*, is said to have additive multiplicative overhead if* $|\mathsf{TwoMsgABE}.sk_M| = \cdot|M| + \mathrm{poly}(\lambda)$*, where* $(\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}) \leftarrow \mathsf{TwoMsgABE.Setup}(1^\lambda)$ *and* $\mathsf{TwoMsgABE}.sk_M \leftarrow \mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, M \in \mathcal{M})$*.*

D.2 Construction

We realize this primitive along the same lines as described by Goldwasser et al. [27]. The idea is to have two instantiations of a ABE scheme. To encrypt an attribute x and two messages $(\mathsf{msg}_0, \mathsf{msg}_1)$, we encrypt (x, msg_0) in one instantiation and (x, msg_1) in the other. Even given attribute keys of M with respect to both the instantiations and the two ciphertexts, there will be exactly one of $(\mathsf{msg}_0, \mathsf{msg}_1)$ that is hidden depending on the value of $M(x)$.

We formally give the construction below. The only tool we use in our construction is 1-key ABE for TMs with additive overhead, $\mathsf{1ABE} = (\mathsf{1ABE.Setup}, \mathsf{1ABE.KeyGen}, \mathsf{1ABE.Enc}, \mathsf{1ABE.Dec})$. We denote the associated class of TMs to be $\mathcal{M}$ and the associated message space to be MSG.

$\underline{\mathsf{TwoMsgABE.Setup}(1^\lambda)}$: On input a security parameter λ in unary, execute $\mathsf{1ABE.Setup}$ twice to obtain $(\mathsf{1ABE.PP}_0, \mathsf{1ABE.SK}_0) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$ and $(\mathsf{1ABE.PP}_1, \mathsf{1ABE.SK}_1) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$. Output $\big(\mathsf{TwoMsgABE.PP} = (\mathsf{1ABE.PP}_0, \mathsf{1ABE.PP}_1), \mathsf{TwoMsgABE.SK} = (\mathsf{1ABE.SK}_0, \mathsf{1ABE.SK}_1)\big)$.

$\underline{\mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, M \in \mathcal{M})}$: On input a secret key $\mathsf{TwoMsgABE.SK} = (\mathsf{1ABE.SK}_0, \mathsf{1ABE.SK}_1)$ and a Turing machine $M \in \mathcal{M}$, first compute two ABE keys: $\mathsf{1ABE}.sk_M^0 \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}_0, M \in \mathcal{M})$ and $\mathsf{1ABE}.sk_M^1 \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}_1, \overline{M})$, where $\overline{M}$ (complement of M) on input x outputs $1 - M(x)$ [13]. Output the attribute key, $\mathsf{TwoMsgABE}.sk_M = (\mathsf{1ABE}.sk_M^0, \mathsf{1ABE}.sk_M^1)$.

$\underline{\mathsf{TwoMsgABE.Enc}(\mathsf{TwoMsgABE.PP}, x, \mathsf{msg}_0, \mathsf{msg}_1)}$: On input a public key $\mathsf{TwoMsgABE.PP} = (\mathsf{1ABE.PP}_0, \mathsf{1ABE.PP}_1)$, attribute $x \in \{0,1\}^*$ and messages $(\mathsf{msg}_0, \mathsf{msg}_1) \in \mathsf{MSG}^2$, compute two ciphertexts: $\mathsf{1ABE.CT}_0 \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg}_0)$ and $\mathsf{1ABE.CT}_1 \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg}_1)$. Output the ciphertext, $\mathsf{TwoMsgABE.CT} = (\mathsf{1ABE.CT}_0, \mathsf{1ABE.CT}_1)$.

$\underline{\mathsf{TwoMsgABE.Dec}(\mathsf{TwoMsgABE}.sk_M, \mathsf{TwoMsgABE.CT})}$: On input an attribute key $\mathsf{TwoMsgABE}.sk_M = (\mathsf{TwoMsgABE}.sk_M^0, \mathsf{TwoMsgABE}.sk_M^1)$ and $\mathsf{TwoMsgABE.CT} = (\mathsf{1ABE.CT}_0, \mathsf{1ABE.CT}_1)$, first compute $\mathsf{out}_0 \leftarrow \mathsf{1ABE.Dec}(\mathsf{1ABE}.sk_M^0, \mathsf{1ABE.CT}_0)$ and then compute $\mathsf{out}_1 \leftarrow \mathsf{1ABE.Dec}(\mathsf{1ABE}.sk_M^1, \mathsf{1ABE.CT}_1)$. Let out_b, for some $b \in \{0,1\}$, be such that $\mathsf{out}_b \neq \perp$. Output $\mathsf{out} = \mathsf{out}_b$.

The correctness of the above scheme follows directly from the correctness of the 1-key ABE scheme $\mathsf{1ABE}$.

Size overhead. Suppose $\mathsf{TwoMsgABE}.sk_M = (\mathsf{1ABE}.sk_M^0, \mathsf{1ABE}.sk_M^1)$ is the output of $\mathsf{TwoMsgABE.KeyGen}(\mathsf{TwoMsgABE.SK}, M \in \mathcal{M})$. We have the size of $\mathsf{TwoMsgABE}.sk_M$ to be,

$$|\mathsf{TwoMsgABE}.sk_M| = |\mathsf{1ABE}.sk_M^0| + |\mathsf{1ABE}.sk_M^1| = 2 \cdot |M| + \mathrm{poly}(\lambda)$$

[13]Here we are only considering Turing machines with boolean output.

This shows that $\mathsf{TwoMsgABE}$ satisfies the constant multiplicative overhead property.

A careful reader will notice that when we instantiate $\mathsf{1ABE}$ using the scheme we constructed in Section 3 then we indeed get a $\mathsf{TwoMsgABE}$ scheme with *additive overhead*. Notice that the attribute key for a machine M in $\mathsf{1ABE}$ is of the form (M, aux), where $|\mathsf{aux}| = \text{poly}(\lambda)$. Now, using the above transformation we have a attribute key in $\mathsf{TwoMsgABE}$ to be of the form $(M, \mathsf{aux}, \overline{M}, \mathsf{aux}')$. This key can be compressed to be of the form $(M, \mathsf{aux}, \mathsf{aux}')$ since $\overline{M}$ can be re-derived from M during the evaluation phase. We thus have the following lemma.

Lemma 2. $\mathsf{TwoMsgABE}$ *satisfies additive overhead property.*

Security. The security of the above scheme is essentially the same proof as in Goldwasser et al. [27]. However, for completeness, we present the proof of security.

Theorem 7. *Assuming the (weak-selective) security of* $\mathsf{1ABE}$*, the scheme* $\mathsf{TwoMsgABE}$ *is (weak-selectively) secure.*

Proof. Suppose $\mathsf{TwoMsgABE}$ is not secure. Denote by $\mathcal{A}$ the PPT adversary that breaks the security of $\mathsf{TwoMsgABE}$. We build a reduction $\mathcal{B}$ that breaks the security of $\mathsf{1ABE}$.

$\mathcal{A}$ sends a Turing machine M, attribute x, message pairs $\Big((\mathsf{msg}_{0,0}, \mathsf{msg}_{0,1}), (\mathsf{msg}_{1,0}, \mathsf{msg}_{1,1})\Big)$. Denote the output of $M(x)$ to be c. The reduction checks if $\mathsf{msg}_{0,c} = \mathsf{msg}_{1,c}$. If this condition is not satisfied then $\mathcal{B}$ aborts. If $\mathcal{B}$ has not aborted, it sends the machine M, attribute x and message pair $(\mathsf{msg}_{0,\overline{c}}, \mathsf{msg}_{1,\overline{c}})$ to the challenger of $\mathsf{1ABE}$. In return $\mathcal{B}$ receives the public key $\mathsf{1ABE.PP}_{\overline{c}}$, attribute key $\mathsf{1ABE}.sk_M^{\overline{c}}$ and challenge ciphertext $\mathsf{1ABE.CT}^*_{\overline{c}}$. As a next step, $\mathcal{B}$ first executes $(\mathsf{1ABE.PP}_c, \mathsf{1ABE.SK}_c) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$, then generates $\mathsf{1ABE}.sk_M^c \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M)$ and finally executes $\mathsf{1ABE.CT}^*_c \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}_c, \mathsf{msg}_{0,c})$.
$\mathcal{B}$ sets the two-outcome ABE public key $\mathsf{TwoMsgABE.PP} = (\mathsf{1ABE.PP}_0, \mathsf{1ABE.PP}_1)$, attribute key $\mathsf{TwoMsgABE}.sk_M = (\mathsf{1ABE}.sk_M^0, \mathsf{1ABE}.sk_M^1)$, ciphertext $\mathsf{TwoMsgABE.CT}^* = (\mathsf{1ABE.CT}^*_0, \mathsf{1ABE.CT}^*_1)$. It then sends $(\mathsf{TwoMsgABE.PP}, \mathsf{TwoMsgABE}.sk_M, \mathsf{TwoMsgABE.CT}^*)$ to $\mathcal{A}$. The output of $\mathcal{B}$ is the output of $\mathcal{A}$.

It can be observed that the advantage of $\mathcal{B}$ in the security game of $\mathsf{1ABE}$ is exactly the same as the advantage of $\mathsf{1ABE}$ in the security game of $\mathsf{TwoMsgABE}$. From our hypothesis, the advantage of $\mathsf{1ABE}$ is non-negligible, contradicting the security of $\mathsf{1ABE}$. □

D.3 Proof of Security of OEE

We first focus on proving that the oblivious evaluation encoding scheme satisfies the indistinguishability of encoding bit property and later we deal with the indistinguishability of machine encoding property.

Theorem 8. *The scheme* OEE *satisfies indistinguishability of bit encoding property assuming the weak selective security of* $\mathsf{TwoMsgABE}$ *and security of garbling scheme* GC.

Proof. We first design a series of hybrids. The first hybrid corresponds to the real experiment where the challenger picks a bit b at random. In the last hybrid Hyb_3, the bit b is information theoretically hidden from the adversary. The probability that the adversary guesses b is with probability 1/2. Then by arguing that every two consecutive hybrids are computationally indistinguishable, it follows that the probability that the adversary outputs b is negligibly close to 1/2.

We denote the advantage of the adversary in Hyb_i to be $\mathsf{adv}_{\mathcal{A},i}$.

Hybrid Hyb_1: On receiving the TM pair (M_0, M_1) and input x, the challenger first picks a bit $b \in \{0,1\}$ at random. It then runs the setup $\mathsf{OEE.Setup}(1^\lambda)$ to obtain $\mathsf{OEE.sk}$. It then runs $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ to obtain $(\widetilde{M_0, M_1})$. It then executes $\mathsf{OEE.InpEncode}$

$(\mathsf{OEE.sk}, x, b)$ to obtain $\widetilde{(x,b)}$. The challenger finally runs $\mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x)$ to obtain $\mathsf{OEE.sk}_x$.

The challenger then sends $\{\mathsf{OEE.sk}_x, \widetilde{(x,b)}, \widetilde{(M_0, M_1)}\}$ to the adversary. The output of the hybrid is the output of the adversary.

Hybrid Hyb_2: Unlike the previous hybrid the challenger, for every input position of the garbled circuit includes exactly one wire key in the input encoding.

The challenger on receiving the TM pair (M_0, M_1), input x and bit b, does the following. It picks the bit b at random. It executes $\mathsf{OEE.Setup}(1^\lambda)$ to obtain the secret key $\mathsf{OEE.sk} = (\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$. It then executes $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ to obtain the encoding $\widetilde{(M_0, M_1)} = \widetilde{N}$, where N is a program described in Figure 3. It generates the punctured secret key $\mathsf{OEE.sk}_x = (\mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$ as the output of $\mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x)$. The input encoding $\widetilde{(x,b)}$ is computed by executing the steps below:

- For every $\mathsf{ind}_t \in [\lambda]$, it computes the garbled circuit with its wire keys, $\big(\mathsf{gckt}_{\mathsf{ind}_t}, \{w_{i,0}^{\mathsf{ind}_t}, w_{i,1}^{\mathsf{ind}_t}\}_{i\in[q]}\big) \leftarrow \mathsf{Garble}(1^\lambda, G)$, where G is a circuit that is as defined in the honest input encoding procedure.
- For every $i \in [q]$ and $\mathsf{ind}_t \in [\lambda]$, it sets the message $\mathcal{W}_{i,\mathsf{ind}_t} = (w_{i,0}^{\mathsf{ind}_t}, 0^{\ell_w})$ if $N(x, i, \mathsf{ind}_t) = 0$, otherwise it sets $\mathcal{W}_{i,\mathsf{ind}_t} = (0^{\ell_w}, w_{i,1}^{\mathsf{ind}_t})$, where ℓ_w is the length of the garbled circuit wire keys. It then computes $\widetilde{y}_{i,\mathsf{ind}_t} \leftarrow \mathsf{TwoMsgABE.Enc}\big(\mathsf{TwoMsgABE.PP}, (x, i, \mathsf{ind}_t), \mathcal{W}_{i,\mathsf{ind}_t}\big)$.

The challenger then sets the encoding to be $\widetilde{(x,b)} = \big(\mathsf{TwoMsgABE.PP}, \{\mathsf{gckt}_{\mathsf{ind}_t\in[\lambda]}\}_{\mathsf{ind}_t\in[\lambda]}, \{\widetilde{y}_{i,\mathsf{ind}_t}\}_{i\in[q],\mathsf{ind}_t\in[\lambda]}\big)$. The challenger then sends the tuple $\big(\widetilde{(M_0, M_1)}, \widetilde{(x,b)}, \mathsf{OEE.sk}_x\big)$ to the adversary.

Lemma 3. *Assuming the security of* TwoMsgABE, *we have* $|\mathsf{adv}_{\mathcal{A},1} - \mathsf{adv}_{\mathcal{A},2}| \leq \mathsf{negl}(\lambda)$, *where* negl *is a negligible function.*

Proof. To transition from Hyb_1 to Hyb_2, we change the two-outcome ABE ciphertexts one at a time. Consider the following sequence of intermediate hybrids, $\mathsf{Hyb}_{1.j}$, for $j \in [q\lambda]$. The first hybrid $\mathsf{Hyb}_{1.1}$ is identical to Hyb_1 and the final intermediate hybrid $\mathsf{Hyb}_{1.q\lambda}$ is identical to Hyb_2.

Intermediate hybrid, $\mathsf{Hyb}_{1.j}$, for $1 < j < q\lambda$: This is the same as $\mathsf{Hyb}_{1.j-1}$ except that the ABE ciphertext $\widetilde{y}_{i^*,\mathsf{ind}_t^*}$, where $j = (i^* - 1)\cdot\lambda + \mathsf{ind}_t^*$ with $1 \leq i^* \leq q$ and $1 \leq \mathsf{ind}_t^* \leq \lambda$, is composed as follows: the challenger computes $\widetilde{y}_{i,\mathsf{ind}_t} \leftarrow \mathsf{TwoMsgABE.Enc}(\mathsf{TwoMsgABE.PP}, (x, i^*, \mathsf{ind}_t^*), \mathcal{W}_c)$, where $\mathcal{W}_c$ is defined below. As in the description of Hyb_2, here $(w_{i^*,0}^{\mathsf{ind}_t^*}, w_{i^*,1}^{\mathsf{ind}_t^*})$ denotes the i^{*th} wire keys corresponding to the ind_t^{*th} garbled circuit.

$$\mathcal{W}_c = \begin{cases} (w_{i^*,0}^{\mathsf{ind}_t^*}, \perp) & \text{if } N(x, i^*, \mathsf{ind}_t^*) = 0, \\ \\ (\perp, w_{i^*,1}^{\mathsf{ind}_t^*}) & \text{if } N(x, i^*, \mathsf{ind}_t^*) = 1 \end{cases}$$

The rest of the hybrid is as in $\mathsf{Hyb}_{1.j-1}$.

We thus have the following claim.

Claim 4. *Assuming the security of* TwoMsgABE, *we have* $|\mathsf{adv}_{\mathcal{A},1.j-1} - \mathsf{adv}_{\mathcal{A},1.j}| \leq \mathsf{negl}(\lambda)$ *for every* $1 < j \leq q\lambda$, *where* negl *is a negligible function.*

Hence,

$$|\mathsf{adv}_{\mathcal{A},1} - \mathsf{adv}_{\mathcal{A},2}| = \sum_{j=2}^{q\lambda} |\mathsf{adv}_{\mathcal{A},1.j-1} - \mathsf{adv}_{\mathcal{A},1.j}| \leq \mathsf{negl}(\lambda)$$

□

Hybrid Hyb_3: The challenger now simulates the garbled circuits instead of generating them honestly. As in the previous hybrid, the challenger picks the bit b at random and then generates the secret key $\mathsf{OEE.sk} = (\mathsf{TwoMsgABE.SK}, \mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$, TM encoding (M_0, M_1) and punctured key $\mathsf{OEE.sk}_x$.

For the input encoding procedure, we use a simulated garbling procedure denoted by SimGC. It takes as input $(1^\lambda, |G|, \mathsf{out})$ and outputs a garbling of a circuit of size $|G|$ along with wire keys such that the evaluation of the garbled circuit yields the result out. The input encoding $\widetilde{(x,b)}$ is computed by executing the steps below:

- Let the output of M_b on x be out and let t^* be the amount of time taken for the execution. We note that t^* would also be the time taken by $M_{\overline{b}}$ to execute on x. For every $\mathsf{ind}_t \in [\lambda]$, it sets $\mathsf{out}_{\mathsf{ind}_t} = \mathsf{out}$ if $2^{\mathsf{ind}_t} \geq t^*$, and otherwise $\mathsf{out}_{\mathsf{ind}_t} = \perp$. It then computes the *simulated* garbled circuit along with the wire keys, $\big(\mathsf{SimGC}_{\mathsf{ind}_t}, \{w_i^{\mathsf{ind}_t}\}_{i \in [q]}\big) \leftarrow \mathsf{SimGarble}(1^\lambda, 1^{|G|}, \mathsf{out}_{\mathsf{ind}_t})$, where G is a circuit that is as defined in the honest input encoding procedure.
- It then computes the ABE ciphertexts $\widetilde{y}_{i,\mathsf{ind}_t}$, for every $i \in [q], \mathsf{ind}_t \in [\lambda]$, exactly as in the previous hybrid.

The challenger then sets the encoding to be $\widetilde{(x,b)} = \big(\mathsf{TwoMsgABE.PP}, \{\mathsf{SimGC}_{\mathsf{ind}_t}\}_{\mathsf{ind}_t \in [\lambda]}, \{\widetilde{y}_{i,\mathsf{ind}_t}\}_{i \in [q], \mathsf{ind}_t \in [\lambda]}\big)$. The challenger then sends the tuple $\big(\widetilde{(M_0, M_1)}, \widetilde{(x,b)}, \mathsf{OEE.sk}_x\big)$ to the adversary.

Lemma 4. *Assuming the security of the garbling scheme* GC*, we have* $|\mathsf{adv}_{\mathcal{A},2} - \mathsf{adv}_{\mathcal{A},3}| \leq \mathsf{negl}(\lambda)$*, where* negl *is a negligible function.*

Proof. We consider a sequence of intermediate hybrids where we change one garbled circuit at a time. Consider the following sequence of intermediate hybrids $\mathsf{Hyb}_{2.j}$, for $j \in [\lambda]$. The first hybrid $\mathsf{Hyb}_{2.1}$ is identical to Hyb_2 and the final intermediate hybrid $\mathsf{Hyb}_{2.\lambda}$ is identical to Hyb_3. For $j \in [\lambda]$ and $j > 1$ we define the following sequence of hybrids,

Intermediate hybrid, $\mathsf{Hyb}_{2.j}$: This hybrid is identical to $\mathsf{Hyb}_{2.j-1}$ except in the generation of j^{th} garbled circuit in the encryption algorithm. Suppose t^* be such that $M_b(x)$ takes t^* number of steps. If j is such that $2^j < t^*$ then generate $\big(\mathsf{SimGC}_j, \{w_i^j\}_{i\in[q]}\big) \leftarrow \mathsf{SimGarble}(1^\lambda, |G|, \perp)$. Otherwise, generate $\big(\mathsf{SimGC}_j, \{w_i^j\}_{i\in[q]}\big) \leftarrow \mathsf{SimGarble}(1^\lambda, |G|, M_b(x))$. The rest of the garbled circuits and the two-outcome ABE ciphertexts are generated as in $\mathsf{Hyb}_{2.j-1}$.

We thus have the following claim.

Claim 5. *Assuming the security of the garbling schemes* GC*, we have* $|\mathsf{adv}_{\mathcal{A},2.j-1} - \mathsf{adv}_{\mathcal{A},2.j}| \leq \mathsf{negl}(\lambda)$ *for every* $1 < j \leq \lambda$*, where* negl *is a negligible function.*

We thus have,

$$|\mathsf{adv}_{\mathcal{A},2} - \mathsf{adv}_{\mathcal{A},3}| = \sum_{j=2}^{\lambda} |\mathsf{adv}_{\mathcal{A},2.j-1} - \mathsf{adv}_{\mathcal{A},2.j}| \leq \mathsf{negl}(\lambda)$$

□

The probability that $\mathcal{A}$ outputs b in Hyb_3 is $1/2$ since b is information theoretically hidden. Further from Lemmas 3, 4, we have that $|\mathsf{adv}_{\mathcal{A},1} - \mathsf{adv}_{\mathcal{A},3}| \leq \mathsf{negl}(\lambda)$. Combining these two facts we have, $\mathsf{adv}_{\mathcal{A},1} \leq \mathsf{negl}(\lambda)$, as desired. □

Theorem 9. *The scheme* OEE *satisfies the indistinguishability of machine encoding property assuming the security of* FHE.

Proof. Let $(M_0, M_1) \in \mathcal{M}^2$ be the number of time steps and b the bit sent by the adversary to the challenger. And let $\mathsf{OEE.sk}_b = (\mathsf{TwoMsgABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_b)$ be the punctured key and $\widetilde{N}_{(\mathsf{FHE.CT}^*_{\mathsf{TM}_0}, \mathsf{FHE.CT}^*_{\mathsf{TM}_1})}$ be the TM encoding sent by the challenger to the adversary, where (i) $\mathsf{FHE.CT}_{\mathsf{TM}_{\bar{b}}} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_{\bar{b}}, \mathsf{FHE.CT}_{\mathsf{TM}_{\bar{b}}})$, and (ii) $\mathsf{TM}_{\bar{b}}$ is either $M_{\bar{b}}$ or M_b. From the semantic security of FHE, the adversary cannot distinguish the case when $\mathsf{TM}_{\bar{b}} = M_{\bar{b}}$ from the case when $\mathsf{TM}_{\bar{b}} = M_b$. This completes the proof. □

E Proof of Security of Succinct iO

E.1 Correctness

To evaluate the obfuscated program $\left(\widetilde{(M,M)}, \widetilde{C}\right)$ on an input x, the evaluator first computes $\widetilde{C}(x)$. From the correctness of $\mathsf{i}\mathcal{O}$, it follows that the output of $\widetilde{C}(x) = C_{[K,\mathsf{OEE.sk}]}(x)$. From the definition of $C_{[K,\mathsf{OEE.sk}]}(\cdot)$, the correctness of the puncturable PRF and the correctness of the OEE scheme, it follows that $\widetilde{C}(x) = \widetilde{(x,0)}$. In the second step, the evaluator computes $y \leftarrow \mathsf{OEE.Decode}\left(\widetilde{(M,M)}, \widetilde{(x,0)}\right)$. From the correctness of the OEE scheme, it follows that $y = M(x)$, as required.

E.2 Security

Let $M_0, M_1 \in \mathcal{M}$ such that: (a) M_0 and M_1 are functionally equivalent, (b) $|M_0| = |M_1|$, and (c) for every input x, running time of M_0 is equal to the running time of M_1. Let $N = 2^L$ be the total number of inputs to M_0 and M_1. We will prove that $\mathsf{SuccIO}(M_0)$ and $\mathsf{SuccIO}(M_1)$ are ϵ-indistinguishable, where $\epsilon = \mathsf{adv}_{\mathrm{PMHE}_2} + N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{PMHE}_1}(\lambda))$, ignoring constant multiplicative factors. Since punctured PRF can be based on one-way functions and our construction of OEE is based on one-way functions and $\mathsf{i}\mathcal{O}$ for circuits, we get $\epsilon = N \cdot \mathrm{poly}(\mathsf{adv}_{\mathrm{OWF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda))$. When $\mathsf{adv}_{\mathrm{OWF}}(\lambda)$ and $\mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda)$ are sub-exponentially small, then we obtain $\epsilon = \mathsf{negl}(\lambda)$.

We prove the security of the construction by a hybrid argument. We will consider a sequence of five main hybrids $H_0, \ldots, H_5$ such that H_0 (resp., H_5) denotes the real world experiment where the adversary is given the obfuscated program $\left(\widetilde{(M_0,M_0)}, \widetilde{C}\right)$ (resp., $\left(\widetilde{(M_1,M_1)}, \widetilde{C}\right)$). Next, we describe the hybrids.

Hybrid H_0: Real world experiment where machine M_0 is obfuscated. The adversary is given the obfuscated program $\left(\widetilde{(M_0,M_0)}, \widetilde{C}\right)$.

Hybrid H_1: Same as H_0, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C^1_{[K,\mathsf{OEE.sk}_0]}\right)$, where $\mathsf{OEE.sk}_0 \leftarrow \mathsf{OEE.puncBit}(\mathsf{OEE.sk}, 0)$ and $C^1_{[K,\mathsf{OEE.sk}_0]}$ is the circuit described in Figure 6.

```
                    $C^1_{[K,\mathsf{OEE.sk}_0]}(x)$

1. Compute $r \leftarrow \mathsf{PRF}_K(x\|0)$.
2. Compute $\widetilde{(x,0)} \leftarrow \mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_0, x)$ using randomness $r$.
3. Output $\widetilde{(x,0)}$.
```

Figure 6: Circuit $C^1_{[K,\mathsf{OEE.sk}_0]}$.

Hybrid H_2: Same as H_1, except that we replace the machine encoding $\widetilde{(M_0,M_0)}$ with $\widetilde{(M_0,M_1)}$.

Hybrid H_3: Same as H_2, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C^3_{[K,\mathsf{OEE.sk}_1]}\right)$, where $\mathsf{OEE.sk}_1 \leftarrow \mathsf{OEE.puncBit}(\mathsf{OEE.sk}, 1)$ and $C^3_{[K,\mathsf{OEE.sk}_1]}$ is the circuit described in Figure 7.

$C^3_{[K,\mathsf{OEE.sk}_1]}(x)$

1. Compute $r \leftarrow \mathsf{PRF}_K(x\|1)$.
2. Compute $\widetilde{(x,1)} \leftarrow \mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_1, x)$ using randomness r.
3. Output $\widetilde{(x,1)}$.

Figure 7: Circuit $C^3_{[K,\mathsf{OEE.sk}_1]}$.

Hybrid H_4: Same as H_3, except that we replace the machine encoding $(\widetilde{M_0, M_1})$ with $(\widetilde{M_1, M_1})$.

Hybrid H_5: Same as H_4, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow i\mathcal{O}\left(C_{[K,\mathsf{OEE.sk}]}\right)$ where $C_{[K,\mathsf{OEE.sk}]}$ is the circuit described in Figure 4. This is the real world experiment where machine M_1 is obfuscated.

This completes the description of the main hybrids.

Indistinguishability of H_0 and H_1. We show that the circuits $C_{[K,\mathsf{OEE.sk}]}$ and $C^1_{[K,\mathsf{OEE.sk}_0]}$ are functionally equivalent. The indistinguishability of H_0 and H_1 then follows from the security of the indistinguishability obfuscator $i\mathcal{O}$.

Circuit $C_{[K,\mathsf{OEE.sk}]}$ on input x computes $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 0)$ using randomness $r \leftarrow \mathsf{PRF}_K(x\|0)$ while $C^1_{[K,\mathsf{OEE.sk}_0]}$ computes $\mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_0, x)$ using randomness r. From the correctness of bit puncturing property of the OEE scheme, it follows that $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 0) = \mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_0, x)$. Thus, $C_{[K,\mathsf{OEE.sk}]}$ and $C^1_{[K,\mathsf{OEE.sk}_0]}$ are functionally equivalent.

Indistinguishability of H_1 and H_2. Note that in both H_1 and H_2, only the punctured key $\mathsf{OEE.sk}_0$ is used. Then, the indistinguishability of H_1 and H_2 follows from the indistinguishability of machine encoding property of the OEE scheme.

ϵ'-Indistinguishability of H_2 and H_3. We will prove that the experiments H_2 and H_3 are ϵ'-indistinguishable, where $\epsilon' = N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{i\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{PMHE}_1}(\lambda))$, ignoring constant multiplicative factors.

The proof of this case involves several intermediate hybrids. We describe it in Section E.2.1.

Indistinguishability of H_3 and H_4. Note that in both H_3 and H_4, only the punctured key $\mathsf{OEE.sk}_1$ is used. Then, the indistinguishability of H_3 and H_4 follows from the indistinguishability of machine encoding property of the OEE scheme.

Indistinguishability of H_4 and H_5. The proof of this case follows in the same manner as the proof of indistinguishability of hybrids H_0 and H_1. We omit the details.

Completing the proof. Combining the above claims, it follows that experiments H_0 and H_5 are ϵ-indistinguishable, where $\epsilon = \mathsf{adv}_{\mathrm{PMHE}_2} + N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{i\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{PMHE}_1}(\lambda))$, ignoring constant multiplicative factors.

E.2.1 ϵ'-Indistinguishability of H_2 and H_3

To argue ϵ'-indistinguishability of H_2 and H_3, we will consider N internal hybrids $H_{2:1}, \ldots, H_{2:N}$. Let $x_1, \ldots, x_N$ denote the N inputs to machines M_0 and M_1, sorted in lexicographic order. For notational convenience, we think of H_2 as $H_{2:0}$. Below, we describe hybrid $H_{2:i}$, where $1 \leq i \leq N$.

Hybrid $H_{2:i}$: Same as $H_{2:i-1}$, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow i\mathcal{O}\left(C^{2:i}_{[K,\mathsf{OEE.sk}]}\right)$, where $C^{2:i}_{[K,\mathsf{OEE.sk}]}$ is the circuit described in Figure 8.

$$C^{2:i}_{[K,\mathsf{OEE.sk}]}(x)$$

1. If $x \leq x_i$, then $b = 1$, else $b = 0$.
2. Compute $r \leftarrow \mathsf{PRF}_K(x \| b)$.
3. Compute $\widetilde{(x,b)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$ using randomness r.
4. Output $\widetilde{(x,b)}$.

Figure 8: Circuit $C^{2:i}_{[K,\mathsf{OEE.sk}]}$.

For every $0 \leq i \leq N$, we will argue the indistinguishability of $H_{2:i}$ and $H_{2:i+1}$. (Recall that we denote H_2 as $H_{2:0}$.) To facilitate this, we consider another sequence of intermediate hybrids $H_{2:i:1}, \ldots, H_{2:i:4}$, where $0 \leq i < N$. We describe them below.

Hybrid $H_{2:i:1}$: Same as $H_{2:i}$, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}\right)$, where:

- $K_{x_{i+1}} \leftarrow \mathsf{PRFPunc}(K, x_{i+1})$.
- $\mathsf{OEE.sk}_{x_{i+1}} \leftarrow \mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x_{i+1})$.
- $\widetilde{(x_{i+1},0)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x_{i+1}, 0)$ using randomness $r \leftarrow \mathsf{PRF}(K, x_{i+1} \| 0)$.
- Circuit $C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}$ contains the values $K_{x_{i+1}}$, $\mathsf{OEE.sk}_{x_{i+1}}$ and $\widetilde{(x_{i+1},0)}$ hardwired, and is described in Figure 9.

$$C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}(x)$$

1. If $x = x_{i+1}$, output $\widetilde{(x_{i+1},0)}$.
2. If $x \leq x_i$, then $b = 1$, else $b = 0$.
3. Compute $r \leftarrow \mathsf{PRF}_{K_{x_{i+1}}}(x \| b)$.
4. Compute $\widetilde{(x,b)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}_{x_{i+1}}, x, b)$ using randomness r.
5. Output $\widetilde{(x,b)}$.

Figure 9: Circuit $C^{2:i:1}_{[K,\mathsf{OEE.sk}]}$.

Hybrid $H_{2:i:2}$: Same as $H_{2:i:1}$, except that the hardwired value $\widetilde{(x_{i+1},0)} \leftarrow \mathsf{OEE.InpEncode}$ $(\mathsf{OEE.sk}, x, 0)$ is now computed using true randomness (as opposed to PRF generated randomness).

Hybrid $H_{2:i:3}$: Same as $H_{2:i:2}$, except that we now replace the hardwired value $\widetilde{(x_{i+1},0)}$ with $\widetilde{(x_{i+1},1)}$, where $\widetilde{(x_{i+1},1)} \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 1)$ is computed using true randomness.

Hybrid $H_{2:i:4}$: Same as $H_{2:i:3}$, except that the hardwired value $\widetilde{(x_{i+1},1)} \leftarrow \mathsf{OEE.InpEncode}$ $(\mathsf{OEE.sk}, x, 1)$ is now computed using randomness $r \leftarrow \mathsf{PRF}_K(x_{i+1} \| 1)$.

This completes the description of the intermediate hybrids. For every $0 \leq i < N$, we now make the following indistinguishability claims:

- $H_{2:i} \approx H_{2:i:1}$.
- $H_{2:i:1} \approx H_{2:i:2}$.
- $H_{2:i:2} \approx H_{2:i:3}$.

- $H_{2:i:3} \approx H_{2:i:4}$.
- $H_{2:i:4} \approx H_{2:i+1}$.

In addition to the above, we will also prove that $H_{2:N} \approx H_3$. Finally, we will combine all these claims to argue the indistinguishability of H_2 and H_3.

Indistinguishability of $H_{2:i}$ and $H_{2:i:1}$. We show that the two circuits $C^{2:i}_{[K,\mathsf{OEE.sk}]}$ and $C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}$ are functionally equivalent. The indistinguishability of $H_{2:i}$ and $H_{2:i:1}$ then follows from the security of the indistinguishability obfuscator $\mathsf{i}\mathcal{O}$.

First observe that since the punctured PRF preserves functionality under puncturing and the OEE scheme satisfies correctness of input puncturing property, it follows that the behavior of circuits $C^{2:i}_{[K,\mathsf{OEE.sk}]}$ and $C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}$ is identical on all inputs $x \neq x_{i+1}$. On input x_{i+1}, circuit $C^{2:i}_{[K,\mathsf{OEE.sk}]}$ outputs $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x_{i+1}, 0)$ that is computed using randomness $r \leftarrow \mathsf{PRF}_K(x_{i+1}\|0)$, while circuit $C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}$ outputs the hardwired value $\widetilde{(x_{i+1},0)}$. However, it follows from the description of $H_{2:i:1}$ that $\widetilde{(x_{i+1},0)} = \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x_{i+1}, 0)$ (where randomness r as described above is used). Then, combining the above, we have that $C^{2:i}_{[K,\mathsf{OEE.sk}]}$ and $C^{2:i:1}_{\left[K_{x_{i+1}},\mathsf{OEE.sk}_{x_{i+1}},\widetilde{(x_{i+1},0)}\right]}$ are functionally equivalent.

Indistinguishability of $H_{2:i:1}$ and $H_{2:i:2}$. This follows immediately from the security of the punctured PRF family used in the construction.

Indistinguishability of $H_{2:i:2}$ and $H_{2:i:3}$. Note that in both experiments $H_{2:i:2}$ and $H_{2:i:3}$, only the punctured key $\mathsf{OEE.sk}_{x_{i+1}}$ is used. Then, the indistinguishability of $H_{2:i:2}$ and $H_{2:i:3}$ follows from the indistinguishability of encoding bit property of the OEE scheme.

Indistinguishability of $H_{2:i:3}$ and $H_{2:i:4}$. This follows immediately from the security of the punctured PRF family used in the construction.

Indistinguishability of $H_{2:i:4}$ and $H_{2:i+1}$. This follows in the same manner as the proof of the indistinguishability of hybrids $H_{2:i}$ and $H_{2:i:1}$. We omit the details.

Indistinguishability of $H_{2:N}$ and H_3. Let $C^{2:N}_{[K,\mathsf{OEE.sk}]}$ denote the circuit used in hybrid $H_{2:N}$. We will show that the circuits $C^{2:N}_{[K,\mathsf{OEE.sk}]}$ and $C^{3}_{[K,\mathsf{OEE.sk}_1]}$ are functionally equivalent. The indistinguishability of $H_{2:N}$ and H_3 then follows from the security of the indistinguishability obfuscator $\mathsf{i}\mathcal{O}$.

Circuit $C^{2:N}_{[K,\mathsf{OEE.sk}]}$ on input x computes $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 1)$ using randomness $r \leftarrow \mathsf{PRF}_K(x\|1)$ while $C^{3}_{[K,\mathsf{OEE.sk}_1]}$ computes $\mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_1, x)$ using randomness r. From the correctness of bit puncturing property of the OEE scheme, we have that $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 1) = \mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_1, x)$. Thus, $C^{2:N}_{[K,\mathsf{OEE.sk}]}$ and $C^{3}_{[K,\mathsf{OEE.sk}_1]}$ are functionally equivalent.

Completing the proof of ϵ'-Indistinguishability of H_2 and H_3. Combining the above claims, we can first establish that $H_{2:i}$ and $H_{2:i+1}$ are ϵ''-indistinguishable, where $\epsilon'' = \mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{PMHE}_1}(\lambda)$, ignoring constant multiplicative factors. This is true for every i such that $0 \leq i < N$. Iterating over all values of i, we obtain that $H_{2:0}$ and $H_{2:N}$ are $N \cdot \epsilon''$-indistinguishable. Then, since $H_{2:0}$ is the same as H_2, and $H_{2:N}$ and H_3 are indistinguishable (as proven above), it follows that H_2 and H_3 are ϵ'-indistinguishable, where $\epsilon' = N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{PMHE}_1}(\lambda))$, ignoring constant multiplicative factors. This completes the proof.