Indistinguishability Obfuscation for Turing Machines: Constant Overhead and Amortization

Prabhanjan Ananth* Abhishek Jain† Amit Sahai‡

Abstract

We study the asymptotic efficiency of indistinguishability obfuscation (i$\mathcal{O}$) on two fronts:

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

  In this work, we construct the first i$\mathcal{O}$ scheme for (bounded-input) Turing machines that achieves only a *constant* multiplicative overhead in size. The constant in our scheme is, in fact, 2.

- **Amortization:** Suppose we want to obfuscate an arbitrary polynomial number of (bounded-input) Turing machines $M_1, \ldots, M_n$. We ask whether it is possible to obfuscate $M_1, \ldots, M_n$ using a *single* application of an i$\mathcal{O}$ scheme for a circuit family where the size of any circuit is *independent* of n as well the size of any Turing machine M_i.

  In this work, we resolve this question in the affirmative, obtaining a new bootstrapping theorem for obfuscating arbitrarily many Turing machines.

Our results rely on the existence of sub-exponentially secure i$\mathcal{O}$ for circuits and re-randomizable encryption schemes.

In order to obtain these results, we develop a new template for obfuscating Turing machines that is of independent interest and has recently found application in subsequent work on patchable obfuscation [Ananth et al, EUROCRYPT'17].

*University of California Los Angeles. Email: `prabhanjan@cs.ucla.edu`. This work was partially supported by grant #360584 from the Simons Foundation and the grants listed under Amit Sahai.

†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 1619348, 1228984, 1136174, and 1065276, BSF grant 2012378, 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 authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

1 Introduction

The notion of indistinguishability obfuscation (i$\mathcal{O}$) [BGI$^+$12] guarantees that given two equivalent programs M_0 and M_1, their obfuscations are computationally indistinguishable. The first candidate for general-purpose i$\mathcal{O}$ was given by Garg et al. [GGH$^+$13b]. Since their work, i$\mathcal{O}$ has been used to realize numerous advanced cryptographic tasks, such as functional encryption [GGH$^+$13b], deniable encryption [SW14], software watermarking [CHN$^+$16] and PPAD hardness [BPR15], that previously seemed beyond our reach.

Over the last few years, research on i$\mathcal{O}$ constructions has evolved in two directions. The first line of research concerns with developing i$\mathcal{O}$ candidates with stronger security guarantees and progressively weaker reliance on multilinear maps [GGH13a, CLT13, GGH15, CLT15], with the goal of eventually building it from standard cryptographic assumptions. By now, a large sequence of works [GGH$^+$13b, BGK$^+$14, BR14, PST14, GLSW14, AGIS14, Zim15, SZ14, AB15, AJ15, BV15, AJS15, Lin16, LV16, GMM$^+$16] have investigated this line of research. The works of [LV16, GMM$^+$16] constitute the state of the art in this direction, where [LV16] give a construction of i$\mathcal{O}$ for circuits from a concrete assumption on constant-degree multilinear maps, while [GMM$^+$16] gives an i$\mathcal{O}$ candidate in a weak multilinear map model [MSZ16] that resists all known attacks on multilinear maps [CHL$^+$15, GHMS14, BWZ14, CLT14, CGH$^+$15, MSZ16].

Another line of research concerns with building i$\mathcal{O}$ candidates with improved efficiency in a *generic* manner. The goal here is to develop bootstrapping theorems for i$\mathcal{O}$ that achieve greater efficiency when obfuscating different classes of programs. This started with the work of Garg et al. [GGH$^+$13b], which showed, roughly speaking, that i$\mathcal{O}$ for functions computed by branching programs implies i$\mathcal{O}$ for functions computed by general boolean circuits. While this first bootstrapping theorem achieved i$\mathcal{O}$ for all polynomial-time circuits, it still left open the question of obfuscating natural representations of the original program (for example, Turing machines).

Recently, this question was addressed in multiple works [BGL$^+$15, CHJV15, KLW15] showing that i$\mathcal{O}$ for circuits implies i$\mathcal{O}$ for Turing Machines with bounded-length inputs (see also [CH15, CCC$^+$15, CCHR15, ACC$^+$15] for extensions). Moving to the Turing Machine model yields significant efficiency improvements over the circuit model since the size of a Turing Machine may be much smaller than the corresponding circuit size. Importantly, it also achieves per-input running time, as opposed to incurring worst-case running time that is inherent to the circuit model of computation.

Our Work. In this work, we continue the study of bootstrapping mechanisms for i$\mathcal{O}$ to achieve further qualitative and quantitative gains in the asymptotic efficiency of obfuscation. We note that despite the recent advances, existing mechanisms for general-purpose i$\mathcal{O}$ remain highly inefficient and incur large polynomial overhead in the size of the program being obfuscated. We seek to improve the state of affairs on two fronts:

- *Size Efficiency:* First, we seek to develop obfuscation mechanisms where the size of an obfuscated program incurs only a small overhead in the size of the program being obfuscated.
- *Amortization:* Second, we seek to develop i$\mathcal{O}$ amortization techniques, where a single expensive call to an obfuscation oracle (that obfuscates programs of a priori fixed size) can be used to obfuscate arbitrarily many programs.

We expand on each of our goals below. Below, we restrict our discussion to Turing machine obfuscation, which is the main focus of this work.

I. Size Efficiency of i$\mathcal{O}$. All known mechanisms for i$\mathcal{O}$ yield obfuscated programs of size polynomial in the size of the underlying program and the security parameter, thus incurring a multiplicative overhead of at least the security parameter in the size of the underlying program.[1]

[1]For the case of circuits, the recent work of [BV15] gives an i$\mathcal{O}$ construction where the obfuscated circuit incurs

The works of [BCP14, ABG+13, IPS15] achieve these parameters by relying on (public-coin) differing inputs obfuscation [BGI+12, IPS15]. In contrast, [BGL+15, CHJV15, KLW15, CH15, CCC+15, CCHR15, ACC+15] only rely upon $\mathsf{i}\mathcal{O}$ for circuits; however, these works are restricted to programs with bounded-length inputs, and as such incur overhead of $\mathrm{poly}(\lambda, |M|, L)$, where L is the bound on the input length.

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?

More precisely, we ask 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.[2]

II. $\mathsf{i}\mathcal{O}$ Amortization. We next consider the case of obfuscating *multiple* Turing machines. Since known circuit obfuscation mechanisms are inefficient not just in terms of obfuscation size but also the obfuscation time, we would like to minimize the use of circuit obfuscation for obfuscating multiple Turing machines. We ask the following question:

Is it possible to obfuscate arbitrarily many Turing machines by using a *single* invocation to an $\mathsf{i}\mathcal{O}$ obfuscator for circuits of *a priori fixed* polynomial size?

More precisely, let O be an $\mathsf{i}\mathcal{O}$ obfuscator for circuits of a fixed polynomial size. Then, we want the ability to obfuscate multiple Turing machines $M_1, \ldots, M_n$ for an unbounded polynomial n, by making a single invocation to O. As above, we study this question for Turing machines with an a priori fixed input length bound L, and allow O to depend on L.

Note that a successful resolution of this question will yield an *amortization* phenomenon where arbitrarily many Turing machines can be obfuscated using (relatively) less expensive cryptographic primitives and only a single expensive invocation to a circuit obfuscator.

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 both of our aforementioned questions become moot. This is because 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.

In a recent work, [LPST15] 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

only a constant overhead in the size of the underlying circuit but polynomial dependence on the depth of the circuit. While our focus is on Turing machine obfuscation, our results also yield improvements for circuit obfuscation. We refer the reader to Section 1.3 for a comparison of our results with [BV15].

[2]We observe that using (public-coin) differing input obfuscation, a variant of the construction given by [BCP14, ABG+13, IPS15] where FHE is combined with hybrid encryption, can yield constant multiplicative overhead. However, the plausibility of differing input obfuscation has come under scrutiny [GGHW14, BSW16], and unlike for indistinguishability obfuscation [PST14, GLSW14], 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 [AJ15, BV15]. 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.

reduction) is presently known for such randomized encodings. In particular, in the same work, [LPST15] show that such randomized encodings, in general, do not exist.

1.1 Our Results

I. $\mathsf{i}\mathcal{O}$ with Constant Multiplicative Overhead. Our first result is 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 $2|M| + \mathrm{poly}(\lambda, L)$, where L is the input length bound. Our construction is based on sub-exponentially secure $\mathsf{i}\mathcal{O}$ 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 re-randomizable encryption schemes, 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 $2 \cdot |M| + \mathrm{poly}(\lambda, L)$, where L is an input length bound.*

Re-randomizable encryption schemes can be based on standard assumptions such as DDH and learning with errors.

A New Template for Obfuscating Turing Machines. In order to obtain this result, we develop a new template for obfuscating Turing machines starting from indistinguishability obfuscation for circuits. An obfuscation of Turing machine M in our template comprises of:

- A reusable encoding of M.
- An obfuscated input encoder circuit, that takes as input x and produces an encoding of x. This encoding is then decoded, together with the (reusable) encoding of M, to recover $M(x)$.

Our template exhibits two salient properties: (a) the reusable encoding of M is constructed from standard cryptographic primitives, *without any use of* $\mathsf{i}\mathcal{O}$. (b) The size of the input encoder circuit is *independent* of M. In contrast, prior templates for $\mathsf{i}\mathcal{O}$ for Turing machines comprised of a single obfuscated encoder circuit that contains M hardwired in its description, and therefore depends on the size of M.

We use the above template to reduce the problem of construction of iO for Turing machines with constant multiplicative overhead in size to the problem of constructing reusable TM encodings with constant multiplicative overhead in size. We defer discussion on the security properties associated with the reusable encoding scheme to the technical overview (Section 1.2). As we discuss next, our template enables some new applications.

II. A Bootstrapping Theorem for Obfuscating Multiple Programs. We now state our second result. Using our new template for Turing machine obfuscation, we show how to obfuscate $N = \mathrm{poly}(\lambda)$ Turing machines $M_1, \ldots, M_N$, for any polynomial N, using just a single invocation to an obfuscated circuit where the circuit size is independent of N and the size of each M_i and only depends on the security parameter and an input length bound L for every TM M_i. At a high level, this can be achieved by combining the input encoder circuits corresponding to the N machines into a *single* circuit whose size is independent of N.

Theorem 2 (Informal)**.** *Let $\mathsf{i}\mathcal{O}_{\mathsf{ckt}}$ be an indistinguishability obfuscation for circuits scheme for a class of circuits $\mathcal{C}_{\lambda,L}$. There exists an indistinguishability obfuscation scheme $\mathsf{i}\mathcal{O}_{\mathsf{tm}}$ for Turing machines with input length bound L, where any polynomial number of Turing machines can be simultaneously obfuscated by making a single call to $\mathsf{i}\mathcal{O}_{\mathsf{ckt}}$.*

We emphasize that in order to obtain the above result, we crucially rely on the two aforementioned salient properties of our template. Indeed, it is unclear how to use the prior works [BGL+15, CHJV15, KLW15] to obtain the above result.

We remark that the above bootstrapping theorem, combined with the fact that the reusable TM encodings in our template for TM obfuscation achieve constant overhead in size, implies the following useful corollary:

Corollary 1. *Assuming sub-exponentially secure* $\mathsf{i}\mathcal{O}$ *for general circuits and sub-exponentially secure re-randomizable encryption schemes, there exists an* $\mathsf{i}\mathcal{O}$ *scheme for Turing Machines with bounded input length such that the total size of the obfuscations of* N *Turing machines* $M_1, \ldots, M_N$ *is* $2\Sigma_i|M_i| + \mathrm{poly}(\lambda, L)$*, where* L *is an input length bound.*[3]

We refer the reader to Section 1.2.2 for a brief discussion on how we obtain Theorem 2.

III. Subsequent work: Patchable $\mathsf{i}\mathcal{O}$**.** In a subsequent work [AJS], Ananth et al. show how to use our template to construct patchable $\mathsf{i}\mathcal{O}$. This notion allows for updating obfuscation of a Turing machine M to an obfuscation of another machine M' with the help of patches that are privately-generated. At a high level, the reason why our template finds use in their work is because they effectively reduce the problem of patchable $\mathsf{i}\mathcal{O}$ to building a patchable reusable encoding scheme. We refer the reader to [AJS] for more details.

IV. Other Applications. Our result on $\mathsf{i}\mathcal{O}$ for TMs with constant overhead in size can be applied in many applications of $\mathsf{i}\mathcal{O}$ to achieve commensurate efficiency gains. Below we highlight some of these applications.

Functional Encryption with Constant Overhead. Plugging in our $\mathsf{i}\mathcal{O}$ in the functional encryption (FE) scheme of Waters [Wat14],[4] we obtain an FE scheme for Turing machines where the size of a function key for a Turing machine M with input length bound L is only $c \cdot |M| + \mathrm{poly}(\lambda, L)$ for some constant c. Further, the size of a ciphertext for any message x is only $c' \cdot |x| + \mathrm{poly}(\lambda)$ for some constant c'.[5]

The size of the function keys can be further reduced by leveraging the recent result of [AS16] 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 $c \cdot |M| + \mathrm{poly}(\lambda)$ for some constant c.

Reusable Garbled Turing Machines with Constant Overhead. By applying the transformations of De Caro et al. [CIJ+13] and Goldwasser et al. [GKP+13a] on the above FE scheme, we obtain the *first* construction of Reusable Garbled TM 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 $c \cdot |M| + \mathrm{poly}(\lambda)$, while the encoding size of an input x is $c_1 \cdot |x| + c_2|M(x)| + \mathrm{poly}(\lambda)$ for some constants c, c_1, c_2.

Previously, Boneh et al. [BGG+14] constructed reusable garbled *circuits* with additive overhead in either the circuit encoding size, or the input encoding size (but not both simultaneously).

1.2 Technical Overview

We now provide an overview of the main ideas underlying our results. We start by motivating and explaining our new template for succinct $\mathsf{i}\mathcal{O}$ and then explain how we build succinct $\mathsf{i}\mathcal{O}$ with

[3]Note that, in contrast, a naive (direct) use of Theorem 1 would yield a result where the the total size is $2\Sigma_i|M_i| + N \cdot \mathrm{poly}(\lambda, L)$.

[4][Wat14] 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 [Wat14] already achieves the second property.

constant overhead in size. Next, in Section 1.2.2, we explain how we obtain our bootstrapping theorem for obfuscating arbitrarily many Turing machine.

1.2.1 A New Template for Succinct $\mathsf{i}\mathcal{O}$ and Achieving Constant Overhead

We start by recalling the common template for constructing $\mathsf{i}\mathcal{O}$ for Turing machines (TM) used in all the prior works in the literature [BGL+15, CHJV15, KLW15, CH15, CCC+15]. Similar to these works, we focus on the restricted setting of TMs with inputs of a priori bounded length. For simplicity of discussion, however, we will ignore this restriction in this section.

Prior template for succinct $\mathsf{i}\mathcal{O}$. [BGL+15, CHJV15, KLW15] 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.* The first step is to construct a randomized encoding (RE) [IK00, AIK04, AIK06] for Turing machines using $\mathsf{i}\mathcal{O}$ for circuits.
2. *From RE to $\mathsf{i}\mathcal{O}$.* The second step consists of obfuscating the encoding procedure of RE (constructed in the first step). Very roughly, to obfuscate a machine M, we simply obfuscate a circuit $C_{M,K}$ that has the machine M and a PRF key K hardwired. On any input x, circuit C_M outputs a "fresh" RE of $M(x)$ using randomness $\mathsf{PRF}_K(x)$. To recover $M(x)$, the evaluator simply executes the decoding algorithm of RE.

Following [BGL+15, CHJV15, KLW15], all of the subsequent works on succinct $\mathsf{i}\mathcal{O}$ [CH15, CCC+15] follow the above template.[6]

Shortcomings of the prior template. However, as we discuss now, this template is highly problematic for achieving our goal of $\mathsf{i}\mathcal{O}$ with constant multiplicative overhead in size.

- First, note that since the obfuscation of machine M corresponds to obfuscating a circuit that has machine M hardwired, in order to achieve constant overhead in the size of M, we would require the underlying circuit obfuscator to already satisfy the constant overhead property!
- Furthermore, since the description of circuit C_M includes the encoding procedure of the RE, we would require the RE scheme constructed in the first step to not only achieve constant overhead in size, but also in *encoding time*. In particular, we would require that the *running time* of the RE encode procedure on input (M, x) has only a constant multiplicative overhead in $|M| + |x|$.

  We stress that this is a much more serious issue. 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 [IKOS08] for discussion).

Towards that end, we devise a new template for constructing $\mathsf{i}\mathcal{O}$ for TMs which is more amenable to our goal of $\mathsf{i}\mathcal{O}$ with constant overhead in size.

A new template for succinct $\mathsf{i}\mathcal{O}$: Starting ideas. Our first idea is to modify the above template in such a manner that the obfuscated circuit does not contain machine M anymore. Instead, the machine M is encoded separately. Specifically, in our modified template, obfuscation of a machine M consists of two components:

- An encoding $\widetilde{M}$ of the machine M using an encoding key sk.
- Obfuscation of the "input encoder", i.e., a circuit $C'_{sk,K}$ that has hardwired in its description the encoding key sk and a PRF key K. On any input x, $C'_{sk,K}$ computes an input encoding $\tilde{x}$ using sk and randomness $\mathsf{PRF}_K(x)$.

[6]We note that the above template also works for obfuscating RAM programs if we start with an RE for RAM in the first step.

To evaluate the obfuscation of M on an input x, the evaluator first executes the obfuscated circuit to obtain an encoding $\tilde{x}$. It then decodes $(\widetilde{M}, \tilde{x})$ to obtain $M(x)$.

A few remarks are in order: first, note that the above template requires a *decomposable* RE where a machine M and an input x can be encoded separately using an encoding key sk. Second, the RE scheme must be *reusable*, i.e., given an encoding $\widetilde{M}$ of M and multiple input encodings $\tilde{x}_1, \ldots, \tilde{x}_n$ (computed using the same encoding key sk) for any n, it should be possible to decode $(\widetilde{M}, \tilde{x}_i)$ to obtain $M(x_i)$ for every $i \in [n]$.

It is easy to verify the correctness of the above construction. Now, let us see why this template is more suitable for our goal. Observe that if the (reusable) encoding of M has constant overhead in size, then the obfuscation scheme also achieves the same property. Crucially, we do not need the RE scheme to achieve constant overhead in encoding time.

At this point, it seems that we have reduced the problem of $\mathsf{i}\mathcal{O}$ for TMs with constant size overhead to the problem of reusable RE with constant size overhead. This intuition, unfortunately, turns out to be misleading. The main challenge arises in proving security of the above template. Very briefly, we note that following [GLW14, PST14, GLSW14], prior works on succinct $\mathsf{i}\mathcal{O}$ use a common "input-by-input" proof strategy to argue the security of their construction. Recall that the obfuscation of a TM M in these works corresponds to obfuscation of the circuit $C_{M,K}$ described earlier. Then, in order to implement the "input-by-input" proof strategy, it is necessary that the PRF key K supports *puncturing* [SW14, BW13, BGI14, KPTZ13]. Note, however, that in our new template, obfuscation of M consists of a reusable encoding of M and obfuscation of circuit $C'_{sk,K}$ described above. Then, implementing a similar proof strategy for our template would require the encoding key sk of the reusable RE (that is embedded in the circuit $C'_{sk,K}$) to also support puncturing. However, the standard notion of reusable RE [GKP$^+$13a] does not support key puncturing, and therefore, does not suffice for arguing security of the above construction.

Oblivious Evaluation Encodings. Towards that end, our next idea is to develop an "iO-friendly" notion of reusable RE that we refer to as *oblivious evaluation encodings* (OEE). Our definition of OEE is specifically tailored to facilitate a security proof of the construction discussed above.

In an OEE scheme, instead of encoding a single machine, we allow encoding of two machines M_0 and M_1 together using an encoding key sk. Further, an input x is encoded together with a "choice" bit b using sk. The decode algorithm, on input encodings of (x, b) and (M_0, M_1), outputs $M_b(x)$.[7]

An OEE scheme also comes equipped with two *key puncturing* algorithms:

- *Input puncturing*: On input an encoding key sk and input x, it outputs a punctured encoding key sk_x^{inp}. This punctured key allows for computation of encodings of $(x', 0)$ and $(x', 1)$ for all inputs $x' \neq x$. The security property associated with it is as follows: for any input x, given a machine encoding of (M_0, M_1) s.t. $M_0(x) = M_1(x)$ and a punctured key sk_x^{inp}, no PPT adversary should be able to distinguish between encodings of $(x, 0)$ and $(x, 1)$.
- *(Choice) Bit puncturing*: On input an encoding key sk and bit b, it outputs a punctured encoding key sk_b^{bit}. This punctured key allows for computation of encodings of (x, b) for all x. The security property associated with it is as follows: for any machine pair (M_0, M_1), given a punctured key sk_0^{bit}, no PPT adversary should be able to distinguish encoding of (M_0, M_0) from (M_0, M_1). (The security for punctured key sk_1^{bit} can be defined analogously.)

[7] An informed reader might find some similarities between OEE and oblivious transfer [Rab81, EGL85]. Indeed, the name for our primitive is inspired by oblivious transfer.

Finally, we say that an OEE scheme achieves constant multiplicative overhead in size if the size of machine encoding of any pair (M_0, M_1) is $|M_0| + |M_1| + \mathrm{poly}(\lambda)$. Further, similar to reusable RE, we require that the size of the input encoding of x is $\mathrm{poly}(\lambda, |x|)$ and in particular, independent of the size of $|M_0|$ and $|M_1|$.

$\mathsf{i}\mathcal{O}$ for TMs from OEE. We now describe our modified template for constructing $\mathsf{i}\mathcal{O}$ for TMs where reusable RE is replaced with OEE. An obfuscation of a machine M consists of two components: (a) An OEE TM encoding of (M, M) generated using an OEE secret key sk. (b) Obfuscation of the OEE input encoder, i.e., a circuit $C_{sk,K}$ that on input x outputs an OEE input encoding of $(x, 0)$ using the OEE key sk and randomness generated using the PRF key K. To evaluate the obfuscated machine on an input x, an evaluator first computes encoding of $(x, 0)$ using the obfuscated $C_{sk,K}$ and then decodes $(x, 0)$ and (M, M), using the OEE decode algorithm, to obtain $M(x)$.

To prove security, we need to argue that obfuscations of two equivalent machines M_0 and M_1 are computationally indistinguishable. For the above construction, this effectively boils down to transitioning from a hybrid where we give out a machine encoding of (M_0, M_0) to one where we give out a machine encoding of (M_1, M_1). We achieve this by crucially relying on the security of the key puncturing algorithms. Very roughly, we first use the punctured key sk_0^{bit} to transition from (M_0, M_0) to (M_0, M_1) and then later, we use sk_1^{bit} to transition from (M_0, M_1) to (M_1, M_1). In between these steps, we rely on punctured keys sk_x^{inp}, for every input x (one at a time), to transition from a hybrid where the (obfuscated) input encoder produces encodings corresponding to bit $b = 0$ to one where $b = 1$.

Now, note that if we instantiate the above construction with an OEE scheme that achieves constant overhead in size, then the resulting obfuscation scheme also satisfies the same property *even* if the obfuscation of circuit $C_{sk,K}$ has polynomial overhead in size. Here, note that it is crucial that we require that the size of the OEE input encodings to be independent of $|M_0|$ and $|M_1|$. Thus, in order to achieve our goal of $\mathsf{i}\mathcal{O}$ for TMs with constant size overhead, the remaining puzzle piece is a construction of OEE with constant overhead in size. Our main technical contribution is to provide such a construction.

Construction of OEE: Initial Challenges. A natural approach towards constructing OEE is to start with known constructions of reusable RE for TMs. We note that the only known approach in the literature for constructing reusable RE for TMs is due to [GHRW14]. In their approach, for every input, a "fresh" instance of a single-use RE for TMs [BGL+15, CHJV15, KLW15, CH15, CCC+15, AS16] is computed on the fly. However, as discussed at the beginning of this section, in order to achieve constant overhead in size, such an approach would require that the single-use RE achieves constant overhead in *encoding time*, which is a significantly harder problem. Therefore, this approach is ill suited to our goal.

In light of the above, we start from the (single-use) RE construction of Koppula et al. [KLW15] and use the ingredients developed in their work to build all the properties necessary for an OEE scheme with constant overhead in size. Indeed, the work of KLW forms the basis of all known subsequent constructions of randomized encodings for TMs/ RAMs [CH15, CCC+15, CCHR15, ACC+15] that do not suffer from space bound restrictions (unlike [BGL+15, CHJV15]); therefore, it is a natural starting point for our goal.

We start by recalling the RE construction of KLW. We only provide a simplified description, omitting several technical details.

An RE encoding of (M, x) has two components:

- Authenticated Hash Tree: the first component consists of a verifiable hash tree[8] computed on an encryption of the input tape initialized with TM M. The root of the hash tree is authenticated using a special signature.

[8][KLW15] uses a special hash tree called positional accumulator. For this high-level overview, one can think of it as a "iO-friendly" Merkle hash tree. We refer the reader to the technical sections for further details.

- Obfuscated Next Step Function: The second component is an obfuscated circuit of the next step function of $U_x(\cdot)$, where U_x is a universal TM that takes as input a machine M and produces $M(x)$. The hash key, signing and verification keys and decryption key are hardwired inside this obfuscated circuit. It takes as input an encrypted state, an encrypted symbol on the tape along with a proof of validity that consists of authentication path in the hash tree and the signature on the root. Upon receiving such an input, it first checks input validity using the hash key and the signature verification key. It then decrypts the state and the symbol using the decryption key. Next, it executes the next step of the transition function. It then re-encrypts the new state and the new symbol. Using the old authentication path, it recomputes the new authentication path and a fresh signature on the root. Finally, it outputs the new signed root.

A reader familiar with [KLW15] will notice that in the above discussion, we have flipped the roles of the machine and the input. First, it is easy to see that these two presentations are equivalent. More importantly, this specific presentation is crucial to our construction of OEE because by flipping the roles of the machine and the input, we are able to leverage the inherent asymmetry in the machine encoding and input encoding in KLW. This point will become more clear later in this section.

The security proof of KLW, at a high level, works by authenticating one step of the computation at a time. In particular, this idea is implemented using a recurring hybrid where for any execution step i of $U_x(M)$, the obfuscated circuit *only accepts a unique input and all other inputs are rejected*. This unique input is a function of the parameters associated with the hashing scheme, signature scheme and the encryption scheme. We call such hybrids *unique-input accepting hybrids*. Such hybrids have, in fact, been used in other $\mathsf{i}\mathcal{O}$-based constructions as well.[9]

Using the above template, we discuss initial ideas towards constructing an OEE scheme. In the beginning, we restrict our attention to achieving reusability with constant overhead. Later, we will discuss how to achieve the key puncturing properties later.

Challenge #1: Reusability. A natural first idea to achieve reusability is to have the first component in the above construction to be the machine encoding and the second component to be the input encoding. To argue security, lets consider a simple case when the adversary is given a machine encoding of M and two input encodings of x_1 and x_2. A natural first approach is to argue the security of M on x_1 first and then argue the security of M on x_2. The hope here is that we can reuse the (single-use) security proof of KLW separately for x_1 and x_2. This, however, doesn't work because the unique-accepting hybrids corresponding to input x_1 would be incompatible with the computation of M on x_2 (and vice versa). An alternate idea would be to employ *multi-input accepting hybrids* instead of unique-input accepting hybrids, where the obfuscated next step function, for a given i, accepts a fixed set of multiple inputs and rejects all other inputs. However, this would mean that we can only hardwire an a priori fixed number of values in the obfuscated circuit which would then put a bound on the number of input encodings that can be issued. Hence this direction is also not feasible.

In order to resolve the above difficulty, we modify the above template. For any input x, to generate an input encoding, we generate fresh parameters of the hashing, signature and the encryption schemes. We then generate the obfuscated circuit of the next step function of $U_x(\cdot)$ with all the parameters (including the freshly generated ones) hardwired inside it. The machine encoding of M, however, is computed with respect to parameters that are decided at the OEE setup time. At this point it is not clear why correctness should hold: the parameters associated with encodings of x and M are independently generated.

[9]For example, Hubacek and Wichs [HW15] consider a scenario where the obfuscated circuit accepts pre-images of the hash function as input. In order to use $\mathsf{i}\mathcal{O}$ security, they use a special hash function (SSB hash) that is programmed to accept only one input on a special index.

To address this, we introduce a *translation* mechanism that translates machine encoding of M with respect to one set of parameters into another machine encoding of M w.r.t to a different set of parameters. In more detail, every input encoding will be equipped with a freshly generated translator. A translator, associated with an encoding of x, takes as input a machine encoding of M, computed using the parameters part of OEE setup, checks for validity and outputs a new encoding of M corresponding to the fresh parameters associated with the encoding of x. For now, the translator can be thought of as an obfuscated circuit that has hardwired inside it the old and the new parameters. Later we discuss its actual implementation.

Finally, a word about security. Roughly speaking, due to the use of fresh parameters for every input, we are able to reduce security to the one-input security of KLW.

Challenge #2: Constant Overhead. While the above high level approach tackles reusability, it does not yet suffice for achieving constant multiplicative overhead in the size of the machine encodings. Recall that the machine encoding consists of an encryption of the machine along with the hash tree and a signature on the root. We first observe that the hashing algorithm is public and hence, it is not necessary to include the hash tree as part of the input encoding; instead the input encoding can just consist of an encryption of the machine, root of the hash tree and a signature on it. The decoder can reconstruct the hash tree and proceed as before. We can then use an encryption scheme with constant overhead in size to ensure that the encryption of M only incurs constant overhead in size. Note, however, that such an encryption scheme should also be compatible with hash tree computation over it.

While one might envision constructing such a scheme, a bigger issue is the size of the translator. In fact, the size of the translator, as described above, is polynomial in the input length, which corresponds to the size of the machine encoding. It therefore invalidates the efficiency requirement of OEE on the size of input encodings.

One plausible approach to overcome this problem might be to not refresh the encryption of M and in fact just translate the signature associated with the root of the hash tree into a different signature. This would mean that the decryption key associated with the encryption of M would be common among all the input encodings. However, in the security proof, this conflicts with the unique-input accepting hybrids as discussed earlier.

Construction of OEE: Our Approach. The main reason why the above solution does not work is because the machine M is in an encrypted form. Suppose we instead focus on the weaker goal of achieving *authenticity* without privacy. That is, we guarantee the correctness of computation against dishonest evaluators but not hide the machine. Our crucial observation is that the above high level template sans encryption of the machine is already a candidate solution for achieving this weaker security goal. An astute reader would observe that this setting resembles attribute based encryption [SW05, GPSW06] (ABE). Indeed in order to build OEE, our first step is to build an ABE scheme for TMs where the key size incurs a constant multiplicative overhead in the original machine length. We achieve this goal by using the ideas developed above. We then provide a *generic* reduction to transform such an ABE scheme into an OEE scheme satisfying constant multiplicative overhead in size. We now explain our steps in more detail.

ABE for TMs. Recall that in an ABE scheme, an encryption of an attribute, message pair (x, m) can be decrypted using a secret key corresponding to a machine M to recover m only if $M(x) = 1$. An ABE scheme is said to have a constant multiplicative overhead in size if the size of key of M is $c|M| + \mathrm{poly}(\lambda)$ for a constant c. Here, note that neither x nor M are required to be hidden.

The starting point of our construction of ABE is the *message hiding encoding* (MHE) scheme of KLW. An MHE scheme is effectively a "privacy-free" RE scheme, and therefore, perfectly suited for our goal of constructing an ABE scheme. More concretely, an MHE encoding of a tuple (M, x, msg), where M is a TM and x is an input to M, allows one to recover msg iff $M(x) = 1$. On the efficiency side, computing the encoding of (M, x, msg) should take time independent of the time to compute M on x. The important point to note here is that only msg needs to be hidden from the adversary and in particular, it is not necessary to hide the computation of M on x.

The construction of MHE follows along the same lines as the RE construction of KLW with the crucial difference that the machine M (unlike the RE construction) is not encrypted. Following the above discussion, this has the right template from which we can build our ABE scheme. Using KLW's template and incorporating the ideas we developed earlier, we sketch the construction of ABE below. This is an oversimplified version and several intricate technical details are omitted.

- Generate secret key sk, verification key vk of a special signature scheme (termed as splittable signature scheme in [KLW15]). Generate a hash key hk of a verifiable hash tree. The public key consists of (vk, hk) and the secret key is sk.
- ABE key of a machine M is computed by first computing a hash tree on M using hk. The root of the hash tree rt is then signed using sk to obtain σ. Output (M, σ). Note that $|\sigma| = \text{poly}(\lambda)$ and thus, the constant multiplicative overhead property is satisfied.
- ABE encryption of (x, msg) is computed by first computing an obfuscated circuit of the next step function of $U_{x,\mathsf{msg}}(\cdot)$. We have $U_{x,\mathsf{msg}}$ to be a circuit that takes as input circuit C and outputs msg if $C(x) = 1$. The parameters hardwired in this obfuscated circuit contains (among other parameters) (sk', vk') of a splittable signature scheme where (sk', vk') is sampled afresh. In addition, it consists of a *signature translator* $\mathsf{SignProg}$, that we introduced earlier. This signature translator takes as input a pair (rt, σ). Upon receiving such an input, it first verifies the validity of the signature w.r.t vk and then outputs a signature on rt w.r.t sk' if the verification succeeds. Otherwise it outputs $\perp$.
  The final output of the encryption algorithm is the obfuscated circuit along with the signature translator.

To argue security, we need to rely on the underlying security of message hiding encodings. Unlike several recent constructions that use KLW, thanks to the modularization of our approach, we are able to reduce the security of our construction to the security of MHE construction of KLW. We view this as a positive step towards reducing the "page complexity" of research works in this area.

Construction of OEE from ABE for TMs. One of the main differences between OEE and ABE is that OEE guarantees privacy of computation while ABE only offers authenticity. Therefore, we need to employ a privacy mechanism in order to transform ABE into an OEE scheme. A similar scenario was encountered by Goldwasser et al. [GKP+13a] in a different context. Their main goal was to obtain single-key FE for circuits from ABE for circuits while we are interested in constructing OEE, which has seemingly stronger requirements than FE, from ABE for Turing machines. Nevertheless, we show how their techniques will be useful to develop a basic template of our construction of 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 [Gen09]. 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.

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 [GKP+13b]. 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 [NY90]. We encrypt machines M_0 and M_1 using two different FHE public keys. The choice bit puncturing key is set to be one of the FHE secret keys depending on which bit needs to be punctured.

1.2.2 Boostrapping Theorem

We now explain how our template for Turing machine obfuscation can be used to obtain Theorem 2. Suppose that we wish to obfuscate N Turing machines $M_1, \ldots, M_N$ for $N = \text{poly}(\lambda)$.

Using our template discussed above, a starting idea towards obtaining Theorem 2 is as follows. Let K_1 and K_2 be keys for two puncturable PRF families. The obfuscation of $M_1, \ldots, M_N$ consists of the following parts:

- N different OEE TM encodings $(\widetilde{M_1, M_1}), \ldots, (\widetilde{M_N, M_N})$ where each $(\widetilde{M_i, M_i})$ is computed using an encoding key sk_i that is generated using randomness $PRF_{K_1}(i)$.
- Obfuscation of a "joint" input encoder circuit C_{K_1,K_2} that contains K_1 and K_2 hardwired in its description. It takes as input a pair (x, i) and performs the following steps: (a) Compute an OEE encoding key sk_i "on-the-fly" by running the OEE setup algorithm using randomness $\mathsf{PRF}_{K_1}(i)$. (b) Compute and output an OEE input encoding $\widetilde{(x, 0)}_i$ for the ith machine using the key sk_i and fresh randomness $\mathsf{PRF}_{K_2}(i, x)$.

Then security of the above construction can be argued using a straightforward hybrid argument using the puncturing properties of the PRF.

The above idea, however, does not immediately yield Theorem 2. The problem is that the OEE input encoding $\widetilde{(x, 0)}_i$ itself contains obfuscated programs. Therefore, the circuit C_{K_1,K_2} (described above) itself needs to make queries to a circuit obfuscation scheme.

We resolve the above problem in the following manner. Recall from above that an OEE input encoding in our scheme consists of two components: a garbled circuit and an ABE ciphertext. An ABE ciphertext, in turn, consists of obfuscations of two circuits. Lets refer to these circuits as C_1^{sub} and C_2^{sub}. Then, our idea is to simply "absorb" the functionality of C_1^{sub} and C_2^{sub} within C_{K_1,K_2}. In more detail, we consider a modified input encoder circuit C'_{K_1,K_2} that works in three modes: (a) In mode 1, it takes as input (x, i) and simply outputs the garbled circuit component of the input encoding $\widetilde{(x, 0)}_i$. (b) In mode 2, it takes an input for circuit C_1^{sub} and produces its output. (c) In mode 3, it takes an input for circuit C_2^{sub} and produces its output.

With the above modification, obfuscation of $M_1, \ldots, M_N$ now consists of N different OEE TM encodings $(\widetilde{M_1, M_1}), \ldots, (\widetilde{M_N, M_N})$ and obfuscation of the modified input encoder circuit C'_{K_1,K_2}. Crucially, this process only involves a *single* invocation of the circuit obfuscation scheme for the circuit family $\{C'_{K_1,K_2}\}$, where the size of C'_{K_1,K_2} is independent of N as well as the size of any M_i. This gives us Theorem 2.

1.3 Related Work

In a recent work, [BV15] give a construction of $\mathsf{i}\mathcal{O}$ for circuits where the size of obfuscation of a circuit C with depth d and inputs of length L is $2 \cdot C + \text{poly}(\lambda, d, L)$. Their construction relies on (sub-exponentially secure) $\mathsf{i}\mathcal{O}$ for circuits with polynomial overhead and the learning with errors assumption.

While we focus on the Turing machine model in this work, we note that our construction can be easily downgraded to the case of circuits to obtain an $\mathsf{i}\mathcal{O}$ scheme where the size of obfuscation of a circuit C with inputs of length L is $2 \cdot C + \text{poly}(\lambda, L)$. In particular, it does not grow with the circuit depth beyond the dependence on the circuit size. Our construction requires (sub-exponentially secure) $\mathsf{i}\mathcal{O}$ for circuits with poly overhead and re-randomizable encryption schemes.

2 Preliminaries

We denote the security parameter by λ.

2.1 Turing machines (TMs)

A Turing machine is a tuple $M = \langle Q, \Sigma_{\text{inp}}, \Sigma_{\text{tape}}, \perp, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$, where every element in the tuple is defined as follows: (a) Q is the set of finite states. (b) Σ_{inp} is the set of input symbols. (c) Σ_{tape} is the set of tape symbols. (d) $\perp$ denotes the blank symbol. (e) $\delta : Q \times \Sigma_{\text{tape}} \to Q \times \Sigma_{\text{tape}} \times \{+1, -1\}$ is the transition function. (f) $q_0 \in Q$ is the start state. (g) $q_{\text{acc}} \in Q$ is the accept state. (h) $q_{\text{rej}} \in Q$ 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 [PF79] is stated below.

Theorem 3. *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* $\mathsf{TMtoCKT}$.

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.

2.2 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 [BW13, BGI14, KPTZ13], the GGM construction [GGM86] of PRFs from one-way functions yields puncturable PRFs.

Theorem 4 ([GGM86, BW13, BGI14, KPTZ13])**.** *If μ-secure one-way functions*[10] *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.*

2.3 Garbling schemes

Yao in his seminal work [Yao86, LP09] proposed the notion of garbled circuits as a solution to the problem of secure two party computation. Recently, Bellare et al. [BHR12] formalized this in the form of a primitive called *garbling scheme*. We adopt this notion. The syntax of the garbling scheme, defined below, 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, $\left(\mathsf{gckt}, \{w_{i,0}, w_{i,1}\}_{i\in[n]}\right)$.

- **Garbled circuit evaluation algorithm**, $\mathsf{EvalGC}\left(\mathsf{gckt}, \{w_{i,x_i}\}_{i\in[n]}\right)$: On input the garbled circuit gckt and wire keys $\{w_{i,x_i}\}_{i\in[n]}$ corresponding to 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 $(\mathsf{gckt}, \{w_{i,0}, w_{i,1}\}_{i\in[n]}) \leftarrow \mathsf{Garble}(1^\lambda, C)$.

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 1. *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 $C \in \mathcal{C}_n$, any input $x \in \{0,1\}^n$, the following two distributions are computationally distinguishable.*

1. $\left\{ \mathsf{SimGarble}\left(1^\lambda, \phi(C), C(x)\right) \right\}$, *where $\phi(C)$ denotes the topology of the circuit C.*
2. $\left\{ \left(\mathsf{gckt}, \{w_{i,x_i}\}_{i\in[n]}\right) \right\}$, *where* $(\mathsf{gckt}, \{w_{i,0}, w_{i,1}\}_{i\in[n]}) \leftarrow \mathsf{Garble}(1^\lambda, C)$.

2.4 Fully Homomorphic Encryption for circuits

Another main tool that we use in one of our constructions is a fully homomorphic encryption scheme (FHE).

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.

[10]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 $\mu(\lambda)$.

- **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 $\mathsf{FHE.CT}'$.
- **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 $|m| + \text{poly}(\lambda)$. 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. Such symmetric encryption schemes are known from one-way functions[11].

FHE from iO and Re-Randomizable Encryption. Recently, Canetti et al. [CLTV15] proposed a construction of fully homomorphic encryption from (sub-exponentially secure) iO and re-randomizable encryption schemes. We instantiate the FHE scheme we use in our work using this construction.

2.5 Indistinguishability Obfuscation

The notion of indistinguishability obfuscation (iO), first conceived by Barak et al. [BGI+12], 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 2 (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform PPT algorithm* $i\mathcal{O}$ *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:*

[11]For example (from [Gol09]): encryption of m is $(r, \mathsf{PRF}_K(r) \oplus m)$, where K is the symmetric key.

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

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

- **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 $x \in \{0,1\}^{\mathsf{inp}}$ and $|C_0| = |C_1|$, we have:*

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

We can additionally enforce the size of the obfuscation of a circuit $C \in \mathcal{C}_\lambda$ to be $c \cdot |C| + \mathrm{poly}(\mathsf{inp}, \lambda)$, 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 3 (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,*

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

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 [BGL$^+$15, CHJV15, KLW15], 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 $c \cdot |M| + \mathrm{poly}(\lambda, L)$, where c is a constant and L is the upper bound on the input length. A succinct obfuscation satisfying this property is termed as *succinct iO with constant multiplicative overhead.* We formally define this below.

Definition 4 (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*

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

- **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:*

$$\Big|\Pr\left[D(\lambda, \mathsf{SuccIO}(\lambda, M_0, L)) = 1\right] - \Pr[D(\lambda, \mathsf{SuccIO}(\lambda, M_1, L)) = 1]\Big| \leq \mathsf{negl}(\lambda)$$

- **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 in Size:** *There exists a universal constant c such that for every $\lambda \in \mathbb{N}$, for every $M \in \mathcal{M}_\lambda$, we have $|\mathsf{SuccIO}(\lambda, M, L)| = c \cdot |M| + \mathrm{poly}(\lambda, L)$.*

3 KLW Building Blocks [KLW15]

We recall some notions introduced in the work of Koppula, Lewko, Waters [KLW15]. There are three main building blocks: positional accumulators, splittable signatures and iterators. The following definitions are stated verbatim from [KLW15]. In this section, we only state the essential security properties of these primitives we explicitly use in this work. There are other security properties associated with these primitives - we define them in Appendix A.

I. Positional Accumulators. We give a brief overview of this primitive. It consists of the algorithms (Setup, $\mathsf{EnforceRead}$, $\mathsf{EnforceWrite}$, $\mathsf{PrepRead}$, $\mathsf{PrepWrite}$, $\mathsf{VerifyRead}$, $\mathsf{WriteStore}$, Update) and is associated with message space Msg_λ.

The algorithm Setup generates the accumulator public parameters along with the initial storage value and the initial root value. It helps to think of the storage as being a hash tree and its associated accumulator value being the root and they both are initialized to $\perp$. There are two algorithms that generate "fake" public parameters, namely, $\mathsf{EnforceRead}$ and $\mathsf{EnforceWrite}$. The algorithm $\mathsf{EnforceRead}$ takes as input a special index ind^* along with a sequence of k computational steps $(m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k)$ (symbol-index pairs) and produces fake public parameters along with initialized storage and root values. Later, we will put forth a requirement that any PPT adversary cannot distinguish "real" public parameters (generated by Setup) and "fake" public parameters (generated by $\mathsf{EnforceRead}$). Also we put forth an information theoretic requirement that any storage generated by the "fake" public parameters is such that the accumulator value associated with the storage *determines a unique value at the location* ind^*. The algorithm $\mathsf{EnforceWrite}$ has a similar flavor as $\mathsf{EnforceRead}$ and we describe in the formal definition.

Once the setup algorithm is executed, there are two algorithms that deal with arguing about the correctness of the storage. The first one, $\mathsf{PrepRead}$ takes as input a storage, an index and produces the symbol at the location at that index and an accompanying proof – we later require this proof to be "short" (in particular, independent of the size of storage). $\mathsf{PrepWrite}$ essentially does the same task except that it does not output the symbol at that location – that it only produces the proof (in the formal definition, we call this aux). Another procedure, $\mathsf{VerifyRead}$ then verifies whether the proof produced by $\mathsf{PrepRead}$ is valid or not. The above algorithms help to verify the correctness of storage. But how do we compute the storage? $\mathsf{WriteStore}$ takes as input an old storage along with a new symbol and the location where the new symbol needs to be assigned. It updates the storage appropriately and outputs the new storage. The algorithm Update describes how to "succinctly" update the accumulator by just knowing the public parameters, accumulator value, message symbol, index and auxiliary information aux (produced by $\mathsf{WriteStore}$). Here, "succinctness" refers to the fact that the update time of Update is independent of the size of the storage.

Syntax. A positional accumulator for message space Msg_λ consists of the following algorithms.

- $\mathsf{SetupAcc}(1^\lambda, T) \to (\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0)$: 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 $store_0$.

- $\mathsf{EnforceRead}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k), \mathsf{ind}^*) \to (\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0)$: 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 $T-1$. Let $(\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0) \leftarrow \mathsf{SetupAcc}(1^\lambda, T)$. 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 $\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$).

We require the following correctness properties:

1. For every ind between 0 and $T-1$, $\mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_k, \mathsf{ind})$ returns m_i, π, where i is the largest value in $[k]$ such that $\mathsf{ind}_i = \mathsf{ind}$. If no such value exists, then $m_i = \epsilon$.
2. For any ind, let $(m, \pi) \leftarrow \mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_k, \mathsf{ind})$. Then $\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_k, m, \mathsf{ind}, \pi) = True$.

Efficiency. We require that $|\pi|$ is a fixed polynomial in λ, where $(m, \pi) \leftarrow \mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathsf{ind})$. In particular, $|\pi|$ should be independent of $|store_{in}|$. We similarly, require that $|aux|$ to be a fixed polynomial in λ, where $aux \leftarrow \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathsf{ind})$.

The security properties are defined in Section A.

II. Splittable Signatures. A splittable signature scheme is a deterministic signature scheme consisting of the basic algorithms ($\mathsf{SetupSpl}, \mathsf{SignSpl}, \mathsf{VerSpl}$) (i.e, setup, signing and verification algorithms) as in a standard signature scheme except that $\mathsf{SetupSpl}$, in addition to the standard signing key-verification key pair (SK, VK), also outputs a rejection-verification key $\mathsf{VK}_{\mathsf{rej}}$. $\mathsf{VK}_{\mathsf{rej}}$ is defined to be such that it rejects every message-signature pair – this will be useful in the security proof. In addition to the above algorithms, the scheme is associated with algorithms $\mathsf{SplitSpl}$ and $\mathsf{SignSplAbo}$. The algorithm $\mathsf{SplitSpl}$ takes as input a signing key-message pair (SK, m^*) and outputs a verification key $\mathsf{VK}_{\mathsf{one}}$, an all-but-one signing key $\mathsf{SK}_{\mathsf{abo}}$ and an all-but-one verification key $\mathsf{VK}_{\mathsf{abo}}$. Using $\mathsf{VK}_{\mathsf{one}}$ we can verify whether a signature associated with m^* is valid or not. For any other message, $\mathsf{VK}_{\mathsf{one}}$ is useless – that is, the output of the verification algorithm is 0. Similarly, $\mathsf{VK}_{\mathsf{abo}}$ is used to verify the signatures on all messages except m^*. The key $\mathsf{SK}_{\mathsf{abo}}$ is used to sign all the messages except m^*. To sign using the key $\mathsf{SK}_{\mathsf{abo}}$ we use a special signing algorithm $\mathsf{SignSplAbo}$.

The security properties associated are described at a high level below:

1. $\mathsf{VK}_{\mathsf{rej}}$ *indistinguishability*: $\mathsf{VK}_{\mathsf{rej}}$ is indistinguishable from the standard verification key VK when no signatures are given to the adversary.

2. $\mathsf{VK}_{\mathsf{one}}$ *indistinguishability*: It is computationally hard to distinguish $\mathsf{VK}_{\mathsf{one}}$ and VK even when the adversary is given a signature on m^* (and no other signatures).

3. $\mathsf{VK}_{\mathsf{abo}}$ *indistinguishability*: It is hard to distinguish $\mathsf{VK}_{\mathsf{abo}}$ and VK even when the adversary is given the signing key $\mathsf{SK}_{\mathsf{abo}}$.

4. *Splitting Indistinguishability*: Suppose ($\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\mathsf{rej}}$) be the output of one execution of $\mathsf{SetupSpl}$ of the signature scheme. This security property says that for any message m^* it is hard to distinguish the following: (a) ($\sigma_{\mathsf{one}}, \mathsf{VK}_{\mathsf{one}}, \mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}}$) with ($\mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}}$) derived from ($\mathsf{SK}, \mathsf{VK}$) and, (b) ($\sigma_{\mathsf{one}}, \mathsf{VK}_{\mathsf{one}}, \mathsf{SK}^*_{\mathsf{abo}}, \mathsf{VK}^*_{\mathsf{abo}}$) with ($\mathsf{SK}^*_{\mathsf{abo}}, \mathsf{VK}^*_{\mathsf{abo}}$) derived from ($\mathsf{SK}^*, \mathsf{VK}^*$) which in turn is generated independently of (SK, VK).

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 (σ_{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 $m \in \mathsf{Msg}$, $\mathsf{VerSpl}(\mathsf{VK}_{\mathsf{rej}}, m, \sigma) = 0$.

Security: $\mathsf{VK}_{\mathsf{one}}$ indistinguishability. We describe $\mathsf{VK}_{\mathsf{one}}$ indistinguishability security property at a high level below. The rest of the security properties (that will only implicitly be used in our work) is described in Appendix A.

Definition 5 ($\mathsf{VK}_{\mathsf{one}}$ 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.

<u>**III. Iterators**.</u> This notion is similar in spirit to the notion of accumulators described earlier. While accumulators were used to bind the storage, the role of iterators is to bind the state of the Turing machines. An iterator scheme consists of three algorithms ($\mathsf{SetupItr}, \mathsf{ItrEnforce}, \mathsf{Iterate}$). $\mathsf{SetupItr}$ is used to generate the public parameters. $\mathsf{ItrEnforce}$ is used to generate "fake" parameters, that is indistinguishable from the real parameters. These parameters can then be used to generate iterator values using the algorithm $\mathsf{Iterate}$.

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 $\mathcal{S}_\lambda$ consists of three algorithms $\mathsf{SetupItr}$, $\mathsf{ItrEnforce}$ and $\mathsf{Iterate}$ as 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 $v_0 \in \mathcal{S}_\lambda$.

- $\mathsf{ItrEnforce}(1^\lambda, T, \vec{m} = (m_1, \ldots, m_k))$ 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 $v_0 \in \mathcal{S}_\lambda$.

- $\mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, m)$ 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 $v_{\mathsf{out}} \in \mathcal{S}_\lambda$.

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 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 $1 \leq j \leq k-1$.

Remark 3. *Unlike standard cryptographic primitives, positional iterators (as defined by [KLW15]) does not have any correctness notion associated with it.*

4 Attribute-based Encryption for TMs with Additive Overhead

In an attribute-based encryption (ABE) scheme, a message m can be encrypted together with an attribute x such that an evaluator holding a decryption key corresponding to a predicate P can recover m if and only if $P(x) = 1$. Unlike most prior works on ABE that model predicates as circuits, in this work, following [GKP$^+$13b], we model predicates as Turing machines with inputs of arbitrary length. We only consider the setting where the adversary can receive only one decryption key. We refer to this as 1-key ABE.

Below, we start by providing definition of 1-key ABE for TMs. In Section 4.2, we present our construction and then prove its security in Section 4.3. Finally, in Section 4.4, we extend our 1-key ABE construction to build two-outcome ABE for TMs.

4.1 Definition

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.

- **Setup,** $\mathsf{1ABE.Setup}(1^\lambda)$: On input a security parameter λ in unary, it outputs a public key-secret key pair $(\mathsf{1ABE.PP}, \mathsf{1ABE.SK})$.

- **Key Generation,** $\mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M)$: On input a secret key $\mathsf{1ABE.SK}$ and a TM $M \in \mathcal{M}$, it outputs an ABE key $\mathsf{1ABE}.sk_M$.

- **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 $\mathsf{1ABE.CT}_{(x,\mathsf{msg})}$.

- **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 for 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 $\mathcal{A}$.

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

1. $\mathcal{A}$ sends to Ch a tuple consisting of a Turing machine M, an attribute x and two messages $(\mathsf{msg}_0, \mathsf{msg}_1)$. If $M(x) = 1$ then the experiment is aborted.
2. The challenger Ch replies to $\mathcal{A}$ with the public key, decryption key of M, the challenge ciphertext; $\left(\mathsf{1ABE.PP}, \mathsf{1ABE}.sk_M, \mathsf{1ABE.CT}_{(x,\mathsf{msg}_b)}\right)$, where the values are computed as follows:
   - $(\mathsf{1ABE.PP}, \mathsf{1ABE.SK}) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$,
   - $\mathsf{1ABE}.sk_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M)$
   - $\mathsf{1ABE.CT}_{(x,\mathsf{msg}_b)} \leftarrow \mathsf{1ABE.Enc}(\mathsf{1ABE.PP}, x, \mathsf{msg}_b)$.
3. The experiment terminates when the adversary outputs the bit b'.

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

Definition 6. *A 1-key attribute based encryption for TMs scheme is said to be* **weak-selectively secure** *if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary* $\mathcal{A}$*,*

$$\left|\mathsf{Pr}[0 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{1ABE}}(1^\lambda, 0)] - \mathsf{Pr}[0 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{1ABE}}(1^\lambda, 1)]\right| \leq \mathsf{negl}(\lambda)$$

Remark 4. *Henceforth, we will omit the term "weak-selective" when referring to the security of ABE schemes.*

1-Key Attribute Based Encryption for TMs with Additive Overhead. We say that a 1-key attribute based encryption for TMs scheme achieves additive overhead property if the size of an ABE key for a TM M is only $|M| + \mathrm{poly}(\lambda)$. More formally,

Definition 7. *A 1-key attribute based encryption for TMs scheme,* $\mathsf{1ABE}$*, defined for a class of Turing machines* $\mathcal{M}$*, satisfies additive overhead property if* $|\mathsf{1ABE}.sk_M| = |M| + \mathrm{poly}(\lambda)$*, where* $(\mathsf{1ABE.SK}, \mathsf{1ABE.PP}) \leftarrow \mathsf{1ABE.Setup}(1^\lambda)$ *and* $\mathsf{1ABE}.sk_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M \in \mathcal{M})$.

4.2 Construction of 1-Key ABE

We now present our construction of 1-key ABE for TMs. We begin with a brief overview.

Overview. Our construction uses three main primitives imported from [KLW15] – namely, positional accumulators, splittable signatures and iterators. We refer the reader to Section 3 for the description of the primitives.

The setup first generates the setup of the splittable signatures scheme to yield $(\mathsf{SK_{tm}}, \mathsf{VK_{tm}}, \mathsf{VK_{rej}})$. The rejection-verification key $\mathsf{VK_{rej}}$ will be discarded. $(\mathsf{SK_{tm}}, \mathsf{VK_{tm}})$ will be the master signing key-verification key pair. The setup also generates accumulator and iterator parameters.

The signing key $\mathsf{SK_{tm}}$ will be the ABE secret key and the rest of the parameters form the public key. To generate an ABE key of M, first compute the accumulator storage of M. Then sign the accumulator value of M using $\mathsf{SK_{tm}}$ to obtain σ. Output the values M, σ and accumulator value[12].

An ABE encryption of (x, msg) is an obfuscation of the next step function that computes $U_x(\cdot)$ (universal circuit with x hardcoded in it) one step at a time. Call this obfuscated circuit

[12]In this construction, the key generation also outputs an iterator value.

N. Embedded into this obfuscated circuit are accumulator and iterator parameters, part of the public parameters. In addition, it has a PRF key K_A used to generate fresh splittable signature instantiations. In order for this to be compatible with the master signing key, a signature translator is provided as part of the ciphertext. This translator circuit, which will be obfuscated, takes as input message, a signature verifiable using $\mathsf{VK}_{\mathsf{tm}}$ and produces a new signature with respect to parameters generated using K_A. Call this obfuscated circuit S. The ciphertext consists of (N, S).

Construction. We will use the following primitives in our construction:

1. A puncturable PRF family denoted by F.
2. A storage accumulator scheme based on i$\mathcal{O}$ and one-way functions that was constructed by [KLW15]. We denote it by Acc = ($\mathsf{SetupAcc}$, $\mathsf{EnforceRead}$, $\mathsf{EnforceWrite}$, $\mathsf{PrepRead}$, $\mathsf{PrepWrite}$, $\mathsf{VerifyRead}$, $\mathsf{WriteStore}$, Update). Let Σ_{tape} be the associated message space with accumulated value of size ℓ_{Acc} bits.
3. An iterators scheme denoted by Itr =($\mathsf{SetupItr}$, $\mathsf{ItrEnforce}$, $\mathsf{Iterate}$). Let $\{0,1\}^{2\lambda+\ell_{\mathsf{Acc}}}$ be the associated message space with iterated value of size ℓ_{Itr} bits.
4. A splittable signatures scheme denoted by $\mathsf{SplScheme} = (\mathsf{SetupSpl}, \mathsf{SignSpl}, \mathsf{VerSpl}, \mathsf{SplitSpl}, \mathsf{SignSplAbo})$. Let $\{0,1\}^{\ell_{\mathsf{Itr}}+\ell_{\mathsf{Acc}}+2\lambda}$ be the associated message space.

Our Scheme. We now describe our construction of a 1-key ABE scheme $\mathsf{1ABE} = (\mathsf{1ABE.Setup}, \mathsf{1ABE.KeyGen}, \mathsf{1ABE.Enc}, \mathsf{1ABE.Dec})$ for the Turing machine family $\mathcal{M}$. Without loss of generality, the start state of every Turing machine in $\mathcal{M}$ is denoted by q_0. We denote the message space for the ABE scheme as MSG.

$\underline{\mathsf{1ABE.Setup}(1^\lambda)}$: On input a security parameter λ, it first executes the setup of splittable signatures scheme to compute $(\mathsf{SK}_{\mathsf{tm}}, \mathsf{VK}_{\mathsf{tm}}, \mathsf{VK}_{\mathsf{rej}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$. Next, it executes the setup of the accumulator scheme 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,

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

$\underline{\mathsf{1ABE.KeyGen}(\mathsf{SK}_{\mathsf{tm}}, M \in \mathcal{M})}$: On input a master secret key $\mathsf{1ABE.SK} = (\mathsf{1ABE.PP}, \mathsf{SK}_{\mathsf{tm}})$ and a Turing machine $M \in \mathcal{M}$, it executes the following steps:

1. Parse the public key $\mathsf{1ABE.PP}$ as $(\mathsf{VK}_{\mathsf{tm}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)$.

2. **Initialization of the storage tree**: Let $\ell_{\mathsf{tm}} = |M|$ be the length of the machine M. For $1 \leq j \leq \ell_{\mathsf{tm}}$, compute $\widetilde{store}_j = \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1, M_j)$, $aux_j = \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j-1)$, $\widetilde{w}_j = \mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_{j-1}, M_j, j-1, aux_j)$, where M_j denotes the j^{th} bit of M. Set the root $w_0 = \widetilde{w}_{\ell_{\mathsf{tm}}}$.

3. **Signing the accumulator value**: Generate a signature on the message $(v_0, q_0, w_0, 0)$ by computing $\sigma_0 \leftarrow \mathsf{SignSpl}(\mathsf{SK}_{\mathsf{tm}}, \mu = (v_0, q_0, w_0, 0))$, where q_0 is the start state of M.

It outputs the ABE key $\mathsf{1ABE}.sk_M = (M, w_0, \sigma_{\mathsf{tm}}, v_0, \widetilde{store}_0)$.

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

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

1. Sample 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 $\perp$. Compute an obfuscation of the program NxtMsg described in Figure 1, namely $N \leftarrow i\mathcal{O}(\mathsf{NxtMsg}\{U_x(\cdot), \mathsf{msg}, \mathsf{PP}_{\mathsf{Acc}}, \mathsf{PP}_{\mathsf{Itr}}, K_A\})$. 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. Compute an obfuscation of the program $S \leftarrow (\mathsf{SignProg}\{K_A, \mathsf{VK}_{\mathsf{tm}}\})$ 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 $\mathsf{1ABE.CT} = (N, S)$.

Program NxtMsg

Constants: Turing machine $U_x = \langle Q, \Sigma_{\mathrm{tape}}, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}} \rangle$, message 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 $\mathsf{sym}_{\mathsf{in}} \in \Sigma_{\mathrm{tape}}$, 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_{in}, signature σ_{in}, accumulator proof π, 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 $m_{\mathrm{in}} = (v_{\mathsf{in}}, \mathsf{st}_{\mathsf{in}}, w_{\mathsf{in}}, \mathsf{pos}_{\mathsf{in}})$. If $\mathsf{VerSpl}(\mathsf{VK}_A, m_{\mathrm{in}}, \sigma_{\mathsf{in}}) = 0$ 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 $\mathsf{st}_{\mathsf{out}} = q_{\mathrm{rej}}$ output $\perp$.
   - If $\mathsf{st}_{\mathsf{out}} = q_{\mathrm{acc}}$ output msg.
4. **Updating the accumulator and the iterator values**:
   - Compute $w_{\mathsf{out}} = \mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, w_{\mathsf{in}}, \mathsf{sym}_{\mathsf{out}}, \mathsf{pos}_{\mathsf{in}}, aux)$. 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 $m_{\mathrm{out}} = (v_{\mathsf{out}}, \mathsf{st}_{\mathsf{out}}, w_{\mathsf{out}}, \mathsf{pos}_{\mathsf{out}})$ and $\sigma_{\mathsf{out}} = \mathsf{SignSpl}(\mathsf{SK}'_A, m_{\mathrm{out}})$.
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

$\underline{\mathsf{1ABE.Dec}(\mathsf{1ABE}.sk_M, \mathsf{1ABE.CT})}$: On input the ABE key $\mathsf{1ABE}.sk_M = (M, w_0, \sigma_{\mathsf{tm}}, v_0, \widetilde{store}_0)$

Program $\mathsf{SignProg}$

Constants: PRF key K_A and verification key $\mathsf{VK}_{\mathsf{tm}}$.
Input: Message y and a signature σ_{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 $r_A \leftarrow F(K, 0)$. Execute the setup of splittable signatures scheme to compute $(\mathsf{SK}_0, \mathsf{VK}_0) \leftarrow \mathsf{SetupSpl}(1^\lambda; r_A)$.
3. Compute the signature $\sigma_0 \leftarrow \mathsf{SignSpl}(\mathsf{SK}_0, y)$.
4. Output σ_0.

Figure 2: Program $\mathsf{SignProg}$

and a 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**: Let $\ell_{\mathsf{tm}} = |M|$ be the length of the TM M. For $1 \leq j \leq \ell_{\mathsf{tm}}$, update the storage tree by computing, $\widetilde{store}_j = \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{store}_{j-1}, j - 1, M_j)$. Set $store_0 = \widetilde{store}_{\ell_{\mathsf{tm}}}$.
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 $(\mathsf{sym}_{i-1}, \pi_{i-1}) \leftarrow \mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_{i-1})$.

   (b) Compute the auxiliary value, $aux_{i-1} \leftarrow \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{-1}, \mathsf{pos}_{i-1})$.

   (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, $store_i \leftarrow \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{i-1}, \mathsf{pos}_{i-1}, \mathsf{sym}_{w,i})$.

This completes the description of the scheme. The correctness of the above scheme follows along the same lines as the message hiding encoding 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 an attribute x and $\mathsf{1ABE}.sk_M$ is an ABE key for a 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 $W_{t=i}$.[13] Once we show this, the lemma 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.

[13]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 $W_{t=i}$.

Remark 5. *In the description of Koppula et al., the accumulator and the iterator algorithms also take 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. Let $\mathsf{1ABE}.sk_M = (M, w_0, \sigma_{\mathsf{tm}}, v_0, \widetilde{store}_0)$ be an 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 that $|w_0|$ and $|\widetilde{store}_0|$ simply polynomials in the security parameter λ. The signature σ_{tm} on the message w_0 is also of length polynomial in the security parameter. Lastly, the iterator parameter v_0 is also only polynomial in the security parameter. Thus, the size of $\mathsf{1ABE}.sk_M$ is $|M| + \text{poly}(\lambda)$.

4.3 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 [KLW15]. 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 $(\mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0) \leftarrow \mathsf{SetupAcc}(1^\lambda)$. 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 $N \leftarrow \mathsf{i}\mathcal{O}(\mathsf{NxtMsg}\{U_x(\cdot), \mathsf{msg}, \mathsf{PP}_{\mathsf{Acc}}, \mathsf{PP}_{\mathsf{Itr}}, K_A\})$ 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:

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

[*Remark: The values $\widetilde{w}_0$, $\widetilde{store}_0$ and $\mathsf{PP}_{\mathsf{Itr}}$ are not explicitly given out in the message hiding encodings construction of KLW. But in their specific accumulator construction, $\widetilde{w}_0$ 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 to the proof of security.*]

The following theorem was shown in [KLW15].

Theorem 5 ([KLW15],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 obfuscator $\mathsf{i}\mathcal{O}$, accumulators scheme Acc, iterators scheme Itr and splittable signatures scheme $\mathsf{SplScheme}$.*

We prove the following theorem.

Theorem 6. *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 $\mathsf{SplScheme}$.*

Proof. 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.

In more detail, 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, $(\sigma^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{SK}_{\backslash\mathbf{y}}, \mathsf{VK}_{\backslash\mathbf{y}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}_{\mathsf{tm}}, \mathbf{y} = (v_0, q_0, \mathbf{w}, 0))$. Here, $(\mathsf{SK}_{\backslash\mathbf{y}}, \mathsf{VK}_{\backslash\mathbf{y}})$ denote the signing-verification key pair that verifies all messages except $\mathbf{y}$. Of particular interest to us is $\sigma^{\mathbf{y}}_{\mathsf{tm}}$, which is the (deterministic) signature on $\mathbf{y}$ and $\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}$, which is the verification key that only validates the message-signature pair $(\mathbf{y}, \sigma^{\mathbf{y}}_{\mathsf{tm}})$ and invalidates all other message-signature pairs. It finally sets the public key as $\mathsf{1ABE.PP} = (\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)$.

The challenger then sets $\mathsf{1ABE}.sk_M = (M, \mathbf{w}, \sigma^{\mathbf{y}}_{\mathsf{tm}}, v_0, \widetilde{store}_0)$. 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 5), 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 $(\mathbf{y}, \sigma^{\mathbf{y}}_{\mathsf{tm}})$. Even with this additional information, the verification keys $\mathsf{VK}_{\mathsf{tm}}$ from $\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}$, 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. $\square$

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.

In more detail, the challenger upon receiving a TM M, attribute x and messages 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 $\mathsf{1ABE.PP} = (\mathsf{VK}^{\mathbf{y}}_{\mathsf{tm}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)$ as in Hyb_2. Further, it computes the ABE key of M, namely $\mathsf{1ABE}.sk_M = (M, \mathbf{w}, \sigma^{\mathbf{y}}_{\mathsf{tm}}, v_0, \widetilde{store}_0)$, 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 3. 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 $S \leftarrow iO(\mathsf{SignProg}\{K_A, \mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}\})$ as in Hyb_2 and $S^* \leftarrow iO(\mathsf{HybSgn}\{\mathsf{VK}_{\mathsf{tm}}^{\mathbf{w}}, \sigma_0\})$ 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 $\mathsf{HybSgn}\{\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \sigma_0\}$ are functionally equivalent. Consider the input (y, σ) to both the programs. There are two cases to consider:

- **Case** $(y, \sigma) \neq (\mathbf{y}, \sigma_{\mathsf{tm}}^{\mathbf{y}})$: The program $\mathsf{SignProg}\{K_A, \mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}\}(y, \sigma)$ outputs $\perp$ because (y, σ) is invalid with respect to $\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}$. For the same reason, program $\mathsf{HybSgn}\{\mathsf{VK}_{\mathsf{tm}}^{\mathbf{w}}, \sigma_0\}(y, \sigma)$ also outputs $\perp$.
- **Case** $(y, \sigma) = (\mathbf{y}, \sigma_{\mathsf{tm}}^{\mathbf{y}})$: The program $\mathsf{SignProg}\{K_A, \mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}\}(y, \sigma)$ 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. $\square$

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

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

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

Figure 3: 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 5, 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 5.

$\mathcal{A}$ 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, $(\sigma_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{SK}_{\backslash \mathbf{y}}, \mathsf{VK}_{\backslash \mathbf{y}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}_{\mathsf{tm}}, \mathbf{y} = (v_0, q_0, \mathbf{w}, 0)$. Finally, $\mathcal{B}$ generates the obfuscation of the program HybSgn described in Figure 3, $S^* \leftarrow (\mathsf{HybSgn}\{\mathsf{VK}_{\mathsf{tm}}^{\mathbf{w}}, \sigma_0\})$. The reduction then prepares the ABE public key, attribute key and challenge ciphertext as below:

- The public key is set to be $\mathsf{1ABE.PP} = (\mathsf{VK}_{\mathsf{tm}}^{\mathbf{y}}, \mathsf{PP}_{\mathsf{Acc}}, \widetilde{w}_0, \widetilde{store}_0, \mathsf{PP}_{\mathsf{Itr}}, v_0)$.
- The attribute key of M to be $\mathsf{1ABE}.sk_M = (M, \mathbf{w}, \sigma_{\mathsf{tm}}^{\mathbf{y}}, v_0, \widetilde{store}_0)$.
- 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 5. This completes the proof. □

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. □

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

Corollary 2. *There exists a 1-key ABE for TMs scheme assuming the existence of indistinguishability obfuscators for P/poly and one-way functions.*

4.4 1-Key Two-Outcome ABE for TMs

Goldwasser et al. [GKP+13a] 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 opposed to a single secret message. Depending on the output of the predicate, exactly one of the messages is revealed and the other message remains hidden. That is, given an encryption of a single attribute x and two messages $(\mathsf{msg}_0, \mathsf{msg}_1)$, the decryption algorithm on input an ABE key $\mathsf{TwoABE}.sk_M$, outputs msg_0 if $M(x) = 0$ and msg_1 otherwise. The security guarantee then says that if $M(x) = 0$ (resp., $M(x) = 1$) then the pair $(\mathsf{TwoABE}.sk_M, \mathsf{TwoABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)})$, reveal no information about msg_1 (resp., msg_0).

We adapt their definition to 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 below.

4.4.1 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, ($\mathsf{TwoABE.Setup}$, $\mathsf{TwoABE.KeyGen}$, $\mathsf{TwoABE.Enc}$, $\mathsf{TwoABE.Dec}$). The syntax of the algorithms is given below.

- **Setup,** $\mathsf{TwoABE.Setup}(1^\lambda)$: On input a security parameter λ in unary, it outputs a secret key $\mathsf{TwoABE.SK}$ and public key $\mathsf{TwoABE.PP}$.
- **Key Generation,** $\mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, M \in \mathcal{M})$: On input a secret key $\mathsf{TwoABE.SK}$ and a TM $M \in \mathcal{M}$, it outputs an ABE key $\mathsf{TwoABE.SK}_M$.
- **Encryption,** $\mathsf{TwoABE.Enc}(\mathsf{TwoABE.PP}, x, \mathsf{msg}_0, \mathsf{msg}_1)$: On input the public key $\mathsf{TwoABE.PP}$, 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 $\mathsf{TwoABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)}$.
- **Decryption,** $\mathsf{TwoABE.Dec}(\mathsf{TwoABE.SK}_M, \mathsf{TwoABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)})$: 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 an ABE key for M yields the message msg_0 if $M(x) = 0$, and msg_1 otherwise. Formally, $\mathsf{TwoABE.Dec}(\mathsf{TwoABE.SK}_M, \mathsf{TwoABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)})$ is (always) msg_0 if $M(x) = 0$ or msg_1 if $M(x) = 1$, where

- $(\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}) \leftarrow \mathsf{TwoABE.Setup}(1^\lambda)$,

- $\mathsf{TwoABE.SK}_M \leftarrow \mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, M \in \mathcal{M})$ and
- $\mathsf{TwoABE.CT}_{(x,\mathsf{msg}_0,\mathsf{msg}_1)} \leftarrow \mathsf{TwoABE.Enc}(\mathsf{TwoABE.PP}, x, \mathsf{msg}_0, \mathsf{msg}_1)$.

Security. Similar to 1-key ABE for TMs, we define an indistinguishability based security notion of 1-key two-outcome ABE for TMs. 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 $\mathcal{A}$.

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

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 $\mathsf{TwoABE.PP}$, attribute key $\mathsf{TwoABE.SK}_M$ and challenge ciphertext $\mathsf{TwoABE.CT}_{(x,\mathsf{msg}_{b,0},\mathsf{msg}_{b,1})}$, where:
   - $(\mathsf{TwoABE.PP}, \mathsf{TwoABE.SK}) \leftarrow \mathsf{TwoABE.Setup}(1^\lambda)$,
   - $\mathsf{TwoABE.SK}_M \leftarrow \mathsf{TwoABE.KeyGen}(1^\lambda, M)$,
   - $\mathsf{TwoABE.CT}_{(x,\mathsf{msg}_{b,0},\mathsf{msg}_{b,1})} \leftarrow \mathsf{TwoABE.Enc}(\mathsf{TwoABE.PP}, x, \mathsf{msg}_{b,0}, \mathsf{msg}_{b,1})$.
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 secure if any PPT adversary can guess the challenge bit only with negligible probability.

Definition 8. *A 1-key two-outcome ABE for TMs scheme is said to be secure if there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t. for every PPT adversary* $\mathcal{A}$*:*

$$\left|\mathsf{Pr}[0 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{TwoABE}}(1^\lambda, 0)] - \mathsf{Pr}[0 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{TwoABE}}(1^\lambda, 1)]\right| \leq \mathsf{negl}(\lambda)$$

1-Key Two-Outcome ABE for TMs with Constant Multiplicative Overhead. We say that a 1-Key Two-Outcome ABE for TMs achieves constant multiplicative overhead if the size of a decryption key for Turing machine M is $c \cdot |M| + \text{poly}(\lambda)$ for a constant c. The formal definition is provided below.

Definition 9 (Constant Multiplicative Overhead)**.** *A 1-key two-outcome ABE for TMs scheme,* TwoABE*, defined for a class of Turing machines* $\mathcal{M}$*, is said to achieve constant multiplicative overhead if* $|\mathsf{TwoABE.SK}_M| = c \cdot |M| + \text{poly}(\lambda)$*, for a constant* c*, where* $(\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}) \leftarrow \mathsf{TwoABE.Setup}(1^\lambda)$ *and* $\mathsf{TwoABE.SK}_M \leftarrow \mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, M \in \mathcal{M})$.

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 10 (Additive Overhead)**.** *A 1-key two-outcome ABE for TMs scheme,* TwoABE*, defined for a class of Turing machines* $\mathcal{M}$*, is said to achieve additive overhead if* $|\mathsf{TwoABE.SK}_M| = |M| + \text{poly}(\lambda)$*, where* $(\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}) \leftarrow \mathsf{TwoABE.Setup}(1^\lambda)$ *and* $\mathsf{TwoABE.SK}_M \leftarrow \mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, M \in \mathcal{M})$.

4.4.2 Construction

We realize this primitive along the same lines as described by Goldwasser et al. [GKP+13a]. 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 give the formal construction below. The only tool we use in our construction is a 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.

$\underline{\mathsf{TwoABE.Setup}(1^\lambda)}$: 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 $\big(\mathsf{TwoABE.PP} = (\mathsf{1ABE.PP}_0, \mathsf{1ABE.PP}_1), \mathsf{TwoABE.SK} = (\mathsf{1ABE.SK}_0, \mathsf{1ABE.SK}_1)\big)$.

$\underline{\mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, M \in \mathcal{M})}$: On input a secret key $\mathsf{TwoABE.SK} = (\mathsf{1ABE.SK}_0, \mathsf{1ABE.SK}_1)$ and a Turing machine $M \in \mathcal{M}$, first compute two ABE keys: $\mathsf{1ABE.SK}^0_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}_0, M \in \mathcal{M})$ and $\mathsf{1ABE.SK}^1_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}_1, \overline{M})$, where $\overline{M}$ (complement of M) on input x outputs $1 - M(x)$.[14] Output the attribute key, $\mathsf{TwoABE.SK}_M = (\mathsf{1ABE.SK}^0_M, \mathsf{1ABE.SK}^1_M)$.

$\underline{\mathsf{TwoABE.Enc}(\mathsf{TwoABE.PP}, x, \mathsf{msg}_0, \mathsf{msg}_1)}$: On input a public key $\mathsf{TwoABE.PP} = (\mathsf{1ABE.PP}_0, \mathsf{1ABE.PP}_1)$, 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, $\mathsf{TwoABE.CT} = (\mathsf{1ABE.CT}_0, \mathsf{1ABE.CT}_1)$.

$\underline{\mathsf{TwoABE.Dec}(\mathsf{TwoABE.SK}_M, \mathsf{TwoABE.CT})}$: On input an attribute key $\mathsf{TwoABE.SK}_M = (\mathsf{TwoABE.SK}^0_M, \mathsf{TwoABE.SK}^1_M)$ and $\mathsf{TwoABE.CT} = (\mathsf{1ABE.CT}_0, \mathsf{1ABE.CT}_1)$, first compute $\mathsf{out}_0 \leftarrow \mathsf{1ABE.Dec}(\mathsf{1ABE.SK}^0_M, \mathsf{1ABE.CT}_0)$ and then compute $\mathsf{out}_1 \leftarrow \mathsf{1ABE.Dec}(\mathsf{1ABE.SK}^1_M, \mathsf{1ABE.CT}_1)$. 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 the output of $\mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, M \in \mathcal{M})$ is $\mathsf{TwoABE.SK}_M = (\mathsf{1ABE.SK}^0_M, \mathsf{1ABE.SK}^1_M)$. We have that the size of $\mathsf{TwoABE.SK}_M$ is:

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

This shows that TwoABE satisfies the constant multiplicative overhead property.

A careful reader will notice that when we instantiate $\mathsf{1ABE}$ using the scheme we constructed in Section 4 then we indeed get a TwoABE scheme with *additive overhead*. Notice that the attribute key for a machine M in $\mathsf{1ABE}$ is of the form (M, aux), where $|\mathsf{aux}| = \mathrm{poly}(\lambda)$. Now, using the above transformation we have a attribute key in TwoABE 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. TwoABE *satisfies additive overhead property.*

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

Security. The security of the above scheme is essentially the same proof as in Goldwasser et al. [GKP+13a]. However, for completeness, we present a proof sketch.

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

Proof. Suppose TwoABE is not secure. Denote by $\mathcal{A}$ the PPT adversary that breaks the security of TwoABE. 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 $\mathsf{1ABE.SK}^{\overline{c}}_M$ 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 $\mathsf{1ABE.SK}^c_M \leftarrow \mathsf{1ABE.KeyGen}(\mathsf{1ABE.SK}, M)$ 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 $\mathsf{TwoABE.PP} = (\mathsf{1ABE.PP}_0, \mathsf{1ABE.PP}_1)$, attribute key $\mathsf{TwoABE.SK}_M = (\mathsf{1ABE.SK}^0_M, \mathsf{1ABE.SK}^1_M)$, ciphertext $\mathsf{TwoABE.CT}^* = (\mathsf{1ABE.CT}^*_0, \mathsf{1ABE.CT}^*_1)$. It then sends $(\mathsf{TwoABE.PP}, \mathsf{TwoABE.SK}_M, \mathsf{TwoABE.CT}^*)$ to $\mathcal{A}$. The output of $\mathcal{B}$ is the output of $\mathcal{A}$.

It is easy to see 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 TwoABE. From our hypothesis, the advantage of an attacker for $\mathsf{1ABE}$ is non-negligible, contradicting the security of $\mathsf{1ABE}$. □

5 Oblivious Evaluation Encodings

In this section, we define and construct *oblivious evaluation encodings* (OEE). This is a strengthening of the notion of machine hiding encodings (MHE) introduced in [KLW15]. Very briefly, machine hiding encodings are essentially randomized encodings (RE), except that in MHE, the machine needs to be hidden whereas in RE, the input needs to be hidden. More concretely, an MHE scheme for Turing machines has an encoding procedure that encodes the output of a Turing machine M and an input x. This is coupled with a decode procedure that decodes the output $M(x)$. The main efficiency requirement is that the encoding procedure should be much "simpler" than actually computing M on x. The security guarantee states that the encoding does not reveal anything more than $M(x)$.

We make several changes to the notion of MHE to obtain our definition of OEE. First, we require that the machine and the input can be encoded *separately.* Secondly, the machine encoding takes as input two Turing machines (M_0, M_1) and outputs a joint 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. The decode algorithm on input an encoding of (M_0, M_1) and (x, b), outputs $M_b(x)$. 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 $x' \neq x$.

5.1 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 $b \in \{0,1\}$,

$$\mathsf{OEE.Decode}\Big((\widetilde{M_0, M_1}), (\widetilde{x, b})\Big) = M_b(x),$$

where (i) $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$, (ii) $(\widetilde{M_0, M_1}) \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ and, (iii) $(\widetilde{x, b}) \leftarrow \mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$.

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 (i) $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}\left(1^\lambda\right)$; (ii) $(\widetilde{M_0, M_1}) \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ and, (iii) $(\widetilde{x', b}) \leftarrow \mathsf{OEE.pIEncode}\left(\mathsf{OEE.puncInp}\left(\mathsf{OEE.sk}, x\right), x', b\right)$.

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 (i) $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}\left(1^\lambda\right)$, (ii) $(\widetilde{M_0, M_1}) \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ and, (iii) $(\widetilde{x, b}) \leftarrow \mathsf{OEE.pBEncode}\left(\mathsf{OEE.puncBit}\left(\mathsf{OEE.sk}, b\right), x\right)$.

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 $\mathsf{adv}_{\mathrm{OEE}_1} = \Pr[b' = b] - \frac{1}{2}$.

Definition 11 (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,* $\mathsf{adv}_{\mathrm{OEE}_1} = \mathsf{negl}(\lambda)$.

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) $\mathsf{OEE.sk}_c \leftarrow \mathsf{OEE.puncBit}(\mathsf{OEE.sk}, c)$. 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 12 (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,* $\mathsf{adv}_{\mathrm{OEE}_2} = \mathsf{negl}(\lambda)$.

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 13 (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.*

5.2 Construction

Notation. We denote the class of Turing machines associated with oblivious evaluation encoding to be $\mathcal{M}$. For simplicity of notation, we assume that $\mathcal{M}$ 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.

To construct a oblivious evaluation encoding scheme, we will use the following ingredients.

1. A 1-key two-outcome ABE for TMs scheme defined for a class of Turing machines $\mathcal{M}$, represented by $\mathsf{TwoABE} = (\mathsf{TwoABE.Setup}, \mathsf{TwoABE.TMEncode}, \mathsf{TwoABE.InpEncode}, \mathsf{TwoABE.Decode})$.
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 $\mathsf{GC} = (\mathsf{Garble}, \mathsf{EvalGC})$.

Construction. We denote the oblivious evaluation encoding scheme to be $\mathsf{OEE} = (\mathsf{OEE.Setup}, \mathsf{OEE.InpEncode}, \mathsf{OEE.TMEncode}, \mathsf{OEE.Decode})$ that is equipped with auxiliary algorithms $(\mathsf{OEE.puncInp}, \mathsf{OEE.pIEncode}, \mathsf{OEE.puncBit}, \mathsf{OEE.pBEncode})$. 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.

- Run $\mathsf{TwoABE.Setup}(1^\lambda)$ to obtain a secret key-public parameters pair, $(\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP})$.
- Run $\mathsf{FHE.Setup}(1^\lambda)$ twice to obtain 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 $\mathsf{OEE.sk} = (\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.

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

- Parse $\mathsf{OEE.sk}$ as $(\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.
- Compute FHE encryptions of TMs M_0 and M_1 w.r.t public keys $\mathsf{FHE.pk}_0$ and $\mathsf{FHE.pk}_1$, respectively. That is, compute $\mathsf{FHE.CT}_{M_0} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_0, M_0)$ and $\mathsf{FHE.CT}_{M_1} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_1, M_1)$.
- Compute a TwoABE decryption key $\mathsf{TwoABE.SK}_N \leftarrow \mathsf{TwoABE.KeyGen}(\mathsf{TwoABE.SK}, N)$ for the machine $N = N_{\left(\{\mathsf{FHE.pk}_c, \mathsf{FHE.CT}_{M_c}\}_{c \in \{0,1\}}\right)}$ described in Figure 4.

It outputs the TM encoding $(\widetilde{M_0, M_1}) = \mathsf{TwoABE.SK}_N$.

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

$N_{\left(\{\mathsf{FHE.pk}_c, \mathsf{FHE.CT}_{M_c}\}_{c\in\{0,1\}}\right)}(x, i, \mathsf{ind})$

- Let $U = U_{x,\mathsf{ind}}(\cdot)$ be a universal Turing machine that on input a Turing machine M, outputs $M(x)$ if the computation terminates within 2^{ind} number of steps, otherwise it outputs $\perp$.
- Transform the universal Turing machine U into a circuit using Theorem 3 (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.

Figure 4: Description of program N.

- Parse $\mathsf{OEE.sk}$ as $(\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$.
- For $\mathsf{ind} \in [\lambda]$, compute a garbled circuit along with the wire keys, $\left(\mathsf{gckt}_{\mathsf{ind}}, \{w_{i,0}^{\mathsf{ind}}, w_{i,1}^{\mathsf{ind}}\}_{i\in[q]}\right) \leftarrow \mathsf{Garble}(1^\lambda, G)$, 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 $\mathsf{ind} \in [\lambda]$, compute a TwoABE ciphertext $\mathsf{TwoABE.CT}_{i,\mathsf{ind}} \leftarrow \mathsf{TwoABE.Enc}\Big(\mathsf{TwoABE.PP}, (x, i, \mathsf{ind}), w_{i,0}^{\mathsf{ind}}, w_{i,1}^{\mathsf{ind}}\Big)$ of the message pair $(w_{i,0}^{\mathsf{ind}}, w_{i,1}^{\mathsf{ind}})$ along with attribute (x, i, ind).

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

<u>$\mathsf{OEE.Decode}(\widetilde{(M_0, M_1)}, \widetilde{(x,b)})$</u>: On input a TM encoding $\widetilde{(M_0, M_1)}$ and an input encoding $\widetilde{(x,b)}$, it executes the following steps.

- Parse the TM encoding $\widetilde{(M_0, M_1)} = \mathsf{TwoABE.SK}_N$ and the input encoding $\widetilde{(x,b)} = \big(\mathsf{TwoABE.PP}, \{\mathsf{gckt}\}_{\mathsf{ind}\in[\lambda]}, \{\mathsf{TwoABE.CT}_{i,\mathsf{ind}}\}_{i\in[q],\mathsf{ind}\in[\lambda]}\big)$.
- For every $\mathsf{ind} \in [\lambda]$, do the following:
  1. For every $i \in [q]$, execute the decryption procedure of TwoABE to obtain the wire keys of the garbled circuit, $\widetilde{w}_i^{\mathsf{ind}} \leftarrow \mathsf{TwoABE.Dec}(\mathsf{TwoABE.SK}_N, \mathsf{TwoABE.CT}_{i,\mathsf{ind}})$.
  2. Execute $\mathsf{EvalGC}(\mathsf{gckt}_{\mathsf{ind}}, \widetilde{w}_1^{\mathsf{ind}}, \ldots, \widetilde{w}_q^{\mathsf{ind}})$ to obtain $\mathsf{out}_{\mathsf{ind}}$.
  3. If $\mathsf{out}_{\mathsf{ind}} \neq \perp$ then output $\mathsf{out} = \mathsf{out}_{\mathsf{ind}}$. Otherwise, continue.

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

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

<u>$\mathsf{OEE.pIEncode}(\mathsf{OEE.sk}_x, x')$</u>: On input a 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* $\mathsf{OEE.sk}_x$ *and input* x' *because the master secret key* $\mathsf{TwoABE.SK}$ *is never used during its execution.]*

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

$\underline{\mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_b, x)}$: 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* $\mathsf{OEE.sk}_b$ *and input* x *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, argue that the above scheme satisfies all the properties of an oblivious evaluation encoding scheme.

Correctness. It suffices to argue the correctness of encode and decode property. The other two correctness properties, w.r.t input puncturing and bit puncturing, follow directly from the correctness of encode and decode property: this is because the execution of the algorithms $\mathsf{OEE.pIEncode}$ and $\mathsf{OEE.pBEncode}$ are identical to the execution of $\mathsf{InpEncode}$. We now argue the correctness of encode and decode property below.

Correctness of Encode and Decode: 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 $\mathsf{TwoABE.SK}_N$ be the output of $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ where $N = N_{\left(\{\mathsf{FHE.pk}_c, \mathsf{FHE.CT}_{M_c}\}_{c \in \{0,1\}}\right)}$ and let $\big(\mathsf{TwoABE.PP}, \{\mathsf{gckt}\}_{\mathsf{ind} \in [\lambda]}, \{\mathsf{TwoABE.CT}_{i,\mathsf{ind}}\}_{i \in [q], \mathsf{ind} \in [\lambda]}\big)$ be the output of $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$.

- From the correctness of TwoABE, we have that the output of $\mathsf{TwoABE.Dec}(\mathsf{TwoABE.SK}_N, \mathsf{TwoABE.CT}_{i,\mathsf{ind}})$ is the i^{th} wire key of $\mathsf{gckt}_{\mathsf{ind}}$ that 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} number of steps, under $\mathsf{FHE.pk}_0$ (resp., $\mathsf{FHE.pk}_1$).
- From the correctness of the garbling scheme, it follows that the output of garbled circuit evaluation $\mathsf{EvalGC}(\mathsf{gckt}_{\mathsf{ind}}, \widetilde{w}_1^{\mathsf{ind}}, \ldots, \widetilde{w}_q^{\mathsf{ind}})$ is $M_b(x)$ when $2^{\mathsf{ind}} \geq t^*$ and is $\perp$ otherwise. Since M runs in polynomial time on all inputs, there will exist at least one $\mathsf{ind} \in [\lambda]$ such that $2^{\mathsf{ind}} \geq t^*$.

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. To see this, note that the main bottleneck in the running time of $\mathsf{OEE.Decode}$ is the number of the iterations it executes. The i^{th} iteration takes time polynomial in λ and 2^i. If $\mathsf{ind} \in [\lambda]$ is the smallest number such that $2^{\mathsf{ind}} \geq t^*$, then the number of the iterations in the execution of $\mathsf{OEE.Decode}$ is ind. Thus, the total running time of $\mathsf{OEE.Decode}$ is $(\sum_{j=1}^{\mathsf{ind}} 2^j)\mathrm{poly}(\lambda) = \mathrm{poly}(\lambda, \mathrm{t}^*)$.

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 TwoABE key $\mathsf{TwoABE.SK}_N$ of the program N described in Figure 4. From the additive overhead property of TwoABE, the size of $\mathsf{TwoABE.SK}_N$ is $|N| + \mathrm{poly}(\lambda)$. Also by inspection we have,

$|N| = |M_0| + |M_1| + \text{poly}(\lambda)$ (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 $|M_0| + |M_1| + \text{poly}(\lambda)$.

5.3 Proof of Security of OEE

We first prove that the oblivious evaluation encoding scheme satisfies the indistinguishability of encoding bit property. Later, we prove that it satisfies the indistinguishability of machine encoding property.

Theorem 8. *The scheme* OEE *satisfies indistinguishability of bit encoding property assuming the weak selective security of* TwoABE *and security of garbling scheme* GC.

Proof. We prove security by a hybrid argument. 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. In this case, the advantage of the adversary in guessing b is 0. Then by arguing that every two consecutive hybrids are computationally indistinguishable, it follows that the advantage of the adversary in guessing the correct bit b in the real experiment is negligible.

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

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. Next, it 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 $\widetilde{(M_0, M_1)} = \mathsf{TwoABE.SK}_N$, where N is the program described in Figure 4. Further, it executes $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, b)$ to obtain $\widetilde{(x, b)}$. It finally runs $\mathsf{OEE.puncInp}(\mathsf{OEE.sk}, x)$ to obtain $\mathsf{OEE.sk}_x$.

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

Hybrid Hyb_2: Same as above, except that the challenger, for every input position of the garbled circuit, includes exactly one wire key in the input encoding.

In more detail, the challenger, upon receiving the TM pair (M_0, M_1) and input x, does the following. It picks a bit b at random. It executes $\mathsf{OEE.Setup}(1^\lambda)$ to obtain the secret key $\mathsf{OEE.sk} = (\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$. Next, it executes $\mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M_0, M_1)$ to obtain the encoding $\widetilde{(M_0, M_1)} = \mathsf{TwoABE.SK}_N$, where N is the program described in Figure 4. Further, it generates the punctured secret key $\mathsf{OEE.sk}_x = (\mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$ 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 $\mathsf{ind} \in [\lambda]$, it computes the garbled circuit with its wire keys, $\big(\mathsf{gckt}_{\mathsf{ind}}, \{w_{i,0}^{\mathsf{ind}}, w_{i,1}^{\mathsf{ind}}\}_{i \in [q]}\big) \leftarrow \mathsf{Garble}(1^\lambda, G)$, where G is a circuit that is as defined in the honest input encoding procedure.
- For every $i \in [q]$ and $\mathsf{ind} \in [\lambda]$, it sets the message $\mathcal{W}_{i,\mathsf{ind}} = (w_{i,0}^{\mathsf{ind}}, \perp)$ if $N(x, i, \mathsf{ind}) = 0$, otherwise it sets $\mathcal{W}_{i,\mathsf{ind}} = (\perp, w_{i,1}^{\mathsf{ind}})$. It then computes $\mathsf{TwoABE.CT}_{i,\mathsf{ind}} \leftarrow \mathsf{TwoABE.Enc}\big(\mathsf{TwoABE.PP}, (x, i, \mathsf{ind}), \mathcal{W}_{i,\mathsf{ind}}\big)$.

The challenger sets $\widetilde{(x, b)} = \big(\mathsf{TwoABE.PP}, \{\mathsf{gckt}_{\mathsf{ind} \in [\lambda]}\}_{\mathsf{ind} \in [\lambda]}, \{\mathsf{TwoABE.CT}_{i,\mathsf{ind}}\}_{i \in [q], \mathsf{ind} \in [\lambda]}\big)$. It 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* TwoABE, *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 $\mathsf{Hyb}_{1.j}$: This is the same as $\mathsf{Hyb}_{1.j-1}$ except that the ABE ciphertext $\mathsf{TwoABE.CT}_{i^*,\mathsf{ind}^*}$, where $j = (i^*-1)\cdot\lambda + \mathsf{ind}^*$ with $1 \le i^* \le q$ and $1 \le \mathsf{ind}^* \le \lambda$, is computed as follows: the challenger computes $\mathsf{TwoABE.CT}_{i^*,\mathsf{ind}^*} \leftarrow \mathsf{TwoABE.Enc}(\mathsf{TwoABE.PP}, (x, i^*, \mathsf{ind}^*), \mathcal{W}_c)$, where $\mathcal{W}_c$ is defined below. As in the description of Hyb_2, here $(w^{\mathsf{ind}^*}_{i^*,0}, w^{\mathsf{ind}^*}_{i^*,1})$ denotes the i^{*th} wire keys corresponding to the ind^{*th} garbled circuit.

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

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

We have the following claim.

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

Hence,

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

□

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 $\mathsf{OEE.sk} = (\mathsf{TwoABE.SK}, \mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.sk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_1)$, 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 $\mathsf{ind} \in [\lambda]$, it sets $\mathsf{out}_{\mathsf{ind}} = \mathsf{out}$ if $2^{\mathsf{ind}} \ge t^*$, and otherwise $\mathsf{out}_{\mathsf{ind}} = \perp$. It then computes the *simulated* garbled circuit along with the wire keys, $\big(\mathsf{SimGC}_{\mathsf{ind}}, \{w^{\mathsf{ind}}_i\}_{i\in[q]}\big) \leftarrow \mathsf{SimGarble}(1^\lambda, 1^{|G|}, \mathsf{out}_{\mathsf{ind}})$, where G is a circuit that is as defined in the honest input encoding procedure.
- It then computes the ABE ciphertexts $\mathsf{TwoABE.CT}_{i,\mathsf{ind}}$, for every $i \in [q], \mathsf{ind} \in [\lambda]$, exactly as in the previous hybrid.

The challenger sets $\widetilde{(x,b)} = \big(\mathsf{TwoABE.PP}, \{\mathsf{SimGC}_{\mathsf{ind}}\}_{\mathsf{ind}\in[\lambda]}, \{\mathsf{TwoABE.CT}_{i,\mathsf{ind}}\}_{i\in[q],\mathsf{ind}\in[\lambda]}\big)$. It 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*,* $|\mathsf{adv}_{\mathcal{A},2} - \mathsf{adv}_{\mathcal{A},3}| \le \mathsf{negl}(\lambda)$*, 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. Let t^* be such that $M_b(x)$ takes t^* number of steps. If j is such that $2^j < t^*$ then generate $\left(\mathsf{SimGC}_j, \{w_i^j\}_{i\in[q]}\right) \leftarrow \mathsf{SimGarble}(1^\lambda, |G|, \perp)$. Otherwise, generate $\left(\mathsf{SimGC}_j, \{w_i^j\}_{i\in[q]}\right) \leftarrow \mathsf{SimGarble}(1^\lambda, |G|, M_b(x))$. The rest of the garbled circuits and the TwoABE ciphertexts are generated as in $\mathsf{Hyb}_{2.j-1}$.

We 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 TMs and c be the bit sent by the adversary to the challenger. Let b be a random bit chosen by the challenger. For simplicity, let us assume that $c = 0$. (The proof for the opposite case follows analogously.)

Let $\mathsf{OEE.sk}_b = (\mathsf{TwoABE.PP}, \mathsf{FHE.pk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.sk}_b)$ be the punctured key and $\mathsf{TwoABE.SK}_N$ be the TM encoding sent by the challenger to the adversary, where (a) $N = N_{(\mathsf{FHE.pk}_0, \mathsf{FHE.pk}_1, \mathsf{FHE.CT}^*_{\mathsf{TM}_0}, \mathsf{FHE.CT}^*_{\mathsf{TM}_1})}$, (b) $\mathsf{FHE.CT}_{\mathsf{TM}_0} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_0, \mathsf{FHE.CT}_{\mathsf{TM}_0})$ and $\mathsf{FHE.CT}_{\mathsf{TM}_1} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_1, \mathsf{FHE.CT}_{\mathsf{TM}_1})$, and (c) $\mathsf{TM}_0 = M_0, \mathsf{TM}_1 = M_{1\oplus b}$. From the semantic security of FHE, the adversary cannot distinguish the case when $\mathsf{TM}_1 = M_0$ from the case when $\mathsf{TM}_1 = M_1$. This completes the proof. □

By instantiating fully homomorphic encryptions from sub-exponentially secure iO and re-reandomizable encryption schemes, we have

Theorem 10. *There exists an oblivious evaluation encodings scheme, assuming the existence of sub-exponentially secure iO for circuits and sub-exponentially secure re-randomizable encryption schemes (which can be based on DDH, LWE).*

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

Let $\mathsf{OEE} = (\mathsf{OEE.Setup}, \mathsf{OEE.InpEncode}, \mathsf{OEE.TMEncode}, \mathsf{OEE.Decode})$ be an OEE scheme with constant multiplicative overhead that is equipped with auxiliary algorithms ($\mathsf{OEE.puncInp}, \mathsf{OEE.pIEncode}, \mathsf{OEE.puncBit}, \mathsf{OEE.pBEncode}$). Let $i\mathcal{O}$ 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 constant multiplicative 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:

- $\mathsf{OEE.sk} \leftarrow \mathsf{OEE.Setup}(1^\lambda)$.

- $\widetilde{(M,M)} \leftarrow \mathsf{OEE.TMEncode}(\mathsf{OEE.sk}, M, M)$.
- $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C_{[K,\mathsf{OEE.sk}]}\right)$, where K is a randomly chosen key for the puncturable PRF family and $C_{[K,\mathsf{OEE.sk}]}$ is the circuit described in Figure 5.

$C_{[K,\mathsf{OEE.sk}]}(x)$

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

Figure 5: Circuit $C_{[K,\mathsf{OEE.sk}]}$.

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.

Theorem 11. *If* PRF *is a sub-exponentially secure puncturable PRF,* OEE *is a sub-exponentially secure OEE scheme with constant multiplicative overhead and* $\mathsf{i}\mathcal{O}$ *is a sub-exponentially secure indistinguishability obfuscator for general circuits, then* SuccIO *is a succinct indistinguishability obfuscator with constant multiplicative overhead.*

Below we argue the efficiency of our construction. We deal with the proof of correctness and security later.

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 $|\widetilde{(M,M)}| = 2 \cdot |M| + \text{poly}(\lambda)$. Further, from the efficiency of the $\mathsf{i}\mathcal{O}$ 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 $2 \cdot |M| + \text{poly}(\lambda, L)$, as required.

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.

6.1 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 $\epsilon = \mathsf{adv}_{\mathrm{OEE}_2} + N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{OEE}_1}(\lambda))$, ignoring constant multiplicative factors.[15] Since punctured PRF can be based on one-way functions

[15]Recall that $\mathsf{adv}_{\mathrm{OEE}_2}$ denotes the advantage of an adversary in the indistinguishability of machine encoding experiment for OEE and $\mathsf{adv}_{\mathrm{OEE}_1}$ denotes the advantage of an adversary in the indistinguishability of bit encoding experiment for OEE.

and our construction of OEE is based on one-way functions and $\mathsf{i}\mathcal{O}$ for circuits, we get $\epsilon = N \cdot \mathrm{poly}\left(\mathsf{adv}_{\mathrm{OWF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda)\right)$. 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 $r \leftarrow \mathsf{PRF}_K(x)$.
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 $r \leftarrow \mathsf{PRF}_K(x)$.
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 $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C_{[K,\mathsf{OEE.sk}]}\right)$ where $C_{[K,\mathsf{OEE.sk}]}$ is the circuit described in Figure 5. 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 $\mathsf{i}\mathcal{O}$.

Circuit $C_{[K,\mathsf{OEE.sk}]}$ on input x computes $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 0)$ using randomness $r \leftarrow \mathsf{PRF}_K(x)$ 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 $\epsilon' = N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{OEE}_1}(\lambda))$, ignoring constant multiplicative factors.

The proof of this case involves several intermediate hybrids. We describe it in Section 6.1.1.

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 $\epsilon = \mathsf{adv}_{\mathrm{OEE}_2} + N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{\mathsf{i}\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{OEE}_1}(\lambda))$, ignoring constant multiplicative factors.

6.1.1 ϵ'-Indistinguishability of H_2 and H_3

Let $x_1, \ldots, x_N$ denote the N inputs to machines M_0 and M_1, sorted in lexicographic order. To argue ϵ'-indistinguishability of H_2 and H_3, we will consider $N+1$ internal hybrids $H_{2:0}, \ldots, H_{2:N}$. Below, we describe the hybrids $H_{2:i}$, starting with $i = 0$ and then $0 < i \leq N$.

Hybrid $H_{2:0}$: Same as H_2, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C^{2:0}_{[K,\mathsf{OEE.sk}]}\right)$, where $C^{2:i}_{[K,\mathsf{OEE.sk}]}$ is the same as circuit $C_{[K,\mathsf{OEE.sk}]}$ described in Figure 5.

Hybrid $H_{2:i}$: Same as $H_{2:i-1}$, except that $\widetilde{C}$ is now computed as $\widetilde{C} \leftarrow \mathsf{i}\mathcal{O}\left(C^{2:i}_{[K,\mathsf{OEE.sk}]}\right)$, 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 $r \leftarrow \mathsf{PRF}_K(x)$.
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 $0 \leq i \leq L$, we will argue the indistinguishability of $H_{2:i}$ and $H_{2:i+1}$. To facilitate this, we consider another sequence of intermediate hybrids $H_{2:i:1}, \ldots, H_{2:i:4}$, where $0 \leq i < L$. 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 $r \leftarrow \mathsf{PRF}(K, x_{i+1})$.
- 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 $r \leftarrow \mathsf{PRF}_{K_{x_{i+1}}}(x)$.
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. We now make the following indistinguishability claims:

- $H_2 \approx H_{2:0}$.
- For every $0 \leq i < N$:
  - $H_{2:i} \approx H_{2:i:1}$.
  - $H_{2:i:1} \approx H_{2:i:2}$.
  - $H_{2:i:2} \approx H_{2:i:3}$.
  - $H_{2:i:3} \approx H_{2:i:4}$.
  - $H_{2:i:4} \approx H_{2:i+1}$.
- $H_{2:N} \approx H_2$.

Finally, we will combine all these claims to argue the indistinguishability of H_2 and H_3.

Indistinguishability of H_2 and $H_{2:0}$. Let $C^{2:0}_{[K, \mathsf{OEE.sk}]}$ denote the circuit used in hybrid $H_{2:0}$. We will show that the circuits $C^{2:0}_{[K, \mathsf{OEE.sk}]}$ and $C^{2}_{[K, \mathsf{OEE.sk}_0]}$ are functionally equivalent. The indistinguishability of $H_{2:0}$ and H_2 then follows from the security of the indistinguishability obfuscator $\mathsf{i}\mathcal{O}$.

Circuit $C^{2:0}_{[K, \mathsf{OEE.sk}]}$ on input x computes $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 0)$ using randomness $r \leftarrow \mathsf{PRF}_K(x)$ while $C^{2}_{[K, \mathsf{OEE.sk}_0]}$ computes $\mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_0, x)$ using randomness r. From the correctness of bit puncturing property of the OEE scheme, we have that $\mathsf{OEE.InpEncode}(\mathsf{OEE.sk}, x, 0) = \mathsf{OEE.pBEncode}(\mathsf{OEE.sk}_0, x)$. Thus, $C^{2:0}_{[K,\mathsf{OEE.sk}]}$ and $C^{2}_{[K,\mathsf{OEE.sk}_0]}$ are functionally equivalent.

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 $i\mathcal{O}$.

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 $r \leftarrow \mathsf{PRF}_K(x_{i+1})$, 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 $H_{2:L}$ and H_3. This follows in the same manner as the proof of the indistinguishability of hybrids H_2 and $H_{2:0}$. We omit the details.

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 $\epsilon'' = \mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{i\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{OEE}_1}(\lambda)$, 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 indistinguishable from H_2, and $H_{2:N}$ and H_3 are indistinguishable, it follows that H_2 and H_3 are ϵ'-indistinguishable, where $\epsilon' = N \cdot (\mathsf{adv}_{\mathsf{PRF}}(\lambda) + \mathsf{adv}_{i\mathcal{O}}(\lambda) + \mathsf{adv}_{\mathrm{OEE}_1}(\lambda))$, ignoring constant multiplicative factors. This completes the proof.

By instantiating OEE from sub-exp iO and re-randomizable encryption schemes, we have the following theorem.

Theorem 12. *There exists an succinct iO scheme, assuming the existence of sub-exponentially secure iO for circuits and sub-exponentially secure re-randomizable encryption schemes (which can be based on DDH, LWE).*

References

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, 2015.

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013.

[ACC$^+$15] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating ram computations with adaptive soundness and privacy. Cryptology ePrint Archive, Report 2015/1082, 2015. `http://eprint.iacr.org/`.

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *ACM CCS*, 2014.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 166–175, 2004.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.

[AJS] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Patchable indistinguishability obfuscation: io for evolving software. In *EUROCRYPT*.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015.

[AS16] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *TCC 2016-A*, 2016.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, 2014.

[BGG$^+$14] 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.

[BGI$^+$12] 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.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM*, 2012.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1480–1498, 2015.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *IACR Cryptology ePrint Archive*, 2016.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. *IACR Cryptology ePrint Archive*, 2014:930, 2014.

[CCC$^+$15] 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.

[CCHR15] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Succinct adaptive garbled ram. Cryptology ePrint Archive, Report 2015/1074, 2015. http://eprint.iacr.org/.

[CGH$^+$15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*, 2015.

[CH15] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. *IACR Cryptology ePrint Archive*, 2015:388, 2015.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. In *STOC*, 2015.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, 2015.

[CHN$^+$16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 1115–1127, 2016.

[CIJ$^+$13] 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.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. *IACR Cryptology ePrint Archive*, 2014:975, 2014.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO*, 2015.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *TCC*. 2015.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6), 1985.

[Gen09] 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.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GGH$^+$13b] 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.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 498–527, 2015.

[GGHW14] 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.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. *IACR Cryptology ePrint Archive*, 2014:929, 2014.

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private ram computation. In *FOCS*. IEEE, 2014.

[GKP$^+$13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

[GKP$^+$13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, 2013.

[GLSW14] 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.

[GLW14] Craig Gentry, Allison Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *Advances in Cryptology–CRYPTO 2014*, pages 426–443, 2014.

[GMM$^+$16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshay Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks. In *TCC-B*, 2016.

[Gol09] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge university press, 2009.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 163–172, 2015.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304, 2000.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *ACM STOC*, 2008.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *TCC*, 2015.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, 2015.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 28–57, 2016.

[LP09] 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.

[LPST15] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. *IACR Cryptology ePrint Archive*, 2015:720, 2015.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *FOCS*, 2016.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 629–658, 2016.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, 1990.

[PF79] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, 2014.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 457–473, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *ACM STOC*, 2014.

[SZ14] 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.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014.

[Yao86] 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.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In *EUROCRYPT*, 2015.

A Security Properties of Primitives of [KLW15]

We provide the security properties of the primitives verbatim from [KLW15].

A.1 Security Properties of Positional Accumulators

Let Acc = ($\mathsf{SetupAcc}$, $\mathsf{EnforceRead}$, $\mathsf{EnforceWrite}$, $\mathsf{PrepRead}$, $\mathsf{PrepWrite}$, $\mathsf{VerifyRead}$, $\mathsf{WriteStore}$, Update) be a positional accumulator for symbol set M. We require Acc to satisfy the following notions of security.

Definition 14 (Indistinguishability of Read Setup)**.** *A positional accumulator* Acc *is said to satisfy indistinguishability of read setup if any PPT adversary* $\mathcal{A}$*'s advantage in the security game* $\mathsf{Expt}_{\mathsf{Acc}}(1^\lambda, \mathcal{A})$ *is at most negligible in* λ*, where* $\mathsf{Expt}_{\mathsf{Acc}}$ *is defined as follows.*

$\mathsf{Expt}_{\mathsf{Acc}}(1^\lambda, \mathcal{A})$

1. *Adversary chooses a bound* $T \in \Theta(2^\lambda)$ *and sends it to challenger.*
2. $\mathcal{A}$ *sends* k *messages* $m_1, \ldots, m_k \in \mathsf{M}$ *and* k *indices* $\mathsf{ind}_1, \ldots,$ $indexA_k \in \{0, \ldots, T-1\}$ *to the challenger.*
3. *The challenger chooses a bit* b*. If* $b = 0$*, the challenger outputs* $(\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0) \leftarrow \mathsf{SetupAcc}(1^\lambda, T)$*. Else, it outputs* $(\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0) \leftarrow \mathsf{EnforceRead}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k))$.
4. $\mathcal{A}$ *sends a bit* b'.

$\mathcal{A}$ *wins the security game if* $b = b'$.

Definition 15 (Indistinguishability of Write Setup)**.** *A positional accumulator* Acc *is said to satisfy indistinguishability of write setup if any PPT adversary* $\mathcal{A}$*'s advantage in the security game* $\mathsf{Expt}_{\mathsf{Acc}}(1^\lambda, \mathcal{A})$ *is at most negligible in* λ*, where* $\mathsf{Expt}_{\mathsf{Acc}}$ *is defined as follows.*

$\mathsf{Expt}_{\mathsf{Acc}}(1^\lambda, \mathcal{A})$

1. *Adversary chooses a bound* $T \in \Theta(2^\lambda)$ *and sends it to challenger.*
2. $\mathcal{A}$ *sends* k *messages* $m_1, \ldots, m_k \in \mathsf{M}$ *and* k *indices* $\mathsf{ind}_1, \ldots,$ $indexA_k \in \{0, \ldots, T-1\}$ *to the challenger.*
3. *The challenger chooses a bit* b*. If* $b = 0$*, the challenger outputs* $(\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0) \leftarrow \mathsf{SetupAcc}(1^\lambda, T)$*. Else, it outputs* $(\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0) \leftarrow \mathsf{EnforceWrite}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k))$.
4. $\mathcal{A}$ *sends a bit* b'.

$\mathcal{A}$ *wins the security game if* $b = b'$.

Definition 16 (Read Enforcing)**.** *Consider any* $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $m_1, \ldots, m_k \in \mathsf{M}$, $\mathsf{ind}_1, \ldots, \mathsf{ind}_k \in \{0, \ldots, T-1\}$ *and any* $\mathsf{ind}^* \in \{0, \ldots, T-1\}$.

Let $(\mathsf{PP}_{\mathsf{Acc}}, w_0, \mathsf{st}_0) \leftarrow \mathsf{EnforceRead}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k), \mathsf{ind}^*)$*. 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* $w_j := Update(\mathsf{PP}_{\mathsf{Acc}}, w_{j-1}, m_j, \mathsf{ind}_j, aux_j)$. Acc *is said to be* read enforcing *if* $\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_k, m, \mathsf{ind}^*, \pi) = True$*, then either* $\mathsf{ind}^* \notin \{\mathsf{ind}_1, \ldots, \mathsf{ind}_k\}$ *and* $m = \epsilon$*, or* $m = m_i$ *for the largest* $i \in [k]$ *such that* $\mathsf{ind}_i = \mathsf{ind}^*$*. Note that this is an information-theoretic property: we are requiring that for all other symobls* m*, values of* π *that would cause* $\mathsf{VerifyRead}$ *to output* $True$ *at* ind^* *do no exist.*

Definition 17 (Write Enforcing)**.** *Consider any* $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $m_1, \ldots, m_k \in \mathsf{M}$, $\mathsf{ind}_1, \ldots, \mathsf{ind}_k \in \{0, \ldots, T-1\}$*. Let* $(\mathsf{PP}_{\mathsf{Acc}}, w_0, \mathsf{st}_0) \leftarrow \mathsf{EnforceWrite}(1^\lambda, T, (m_1, \mathsf{ind}_1), \ldots, (m_k, \mathsf{ind}_k))$*. 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 $w_j := Update(\mathsf{PP}_{\mathsf{Acc}}, w_{j-1}, m_j, \mathsf{ind}_j, aux_j)$. Acc is said to be write enforcing *if* $\mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, w_{k-1}, m_k, \mathsf{ind}_k, aux) = w_{out} \neq Reject$, *for any aux, then $w_{out} = w_k$. Note that this is an information-theoretic property: we are requiring that an aux value producing an accumulated value other than w_k or Reject deos not exist.*

A.2 Security Properties of Splittable Signatures

We will now define the security notions for splittable signature schemes. Each security notion is defined in terms of a security game between a challenger and an adversary $\mathcal{A}$.

Definition 18 ($\mathsf{VK}_{\mathsf{rej}}$ indistinguishability)**.** *A splittable signature scheme § is said to be* $\mathsf{VK}_{\mathsf{rej}}$ *indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:*

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

1. *Challenger computes* $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\mathsf{rej}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$ *.Next, it chooses $b \leftarrow \{0,1\}$. If $b = 0$, it sends* VK *to $\mathcal{A}$. Else, it sends* $\mathsf{VK}_{\mathsf{rej}}$.
2. *$\mathcal{A}$ sends its guess b'.*

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

We note that in the game above, $\mathcal{A}$ never receives any signatures and has no ability to produce them. This is why the difference between VK and $\mathsf{VK}_{\mathsf{rej}}$ cannot be tested.

Definition 19 ($\mathsf{VK}_{\mathsf{one}}$ indistinguishability)**.** *A splittable signature scheme § is said to be* $\mathsf{VK}_{\mathsf{one}}$ *indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:*

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

1. *$\mathcal{A}$ sends a message $m^* \in \mathsf{M}$.*
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.

Definition 20 ($\mathsf{VK}_{\mathsf{abo}}$ indistinguishability)**.** *A splittable signature scheme § is said to be* $\mathsf{VK}_{\mathsf{abo}}$ *indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:*

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

1. *$\mathcal{A}$ sends a message $m^* \in \mathsf{M}$.*
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* $(\mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}})$ *to $\mathcal{A}$. Else, it sends* $(\mathsf{SK}_{\mathsf{abo}}, \mathsf{VK})$ *to $\mathcal{A}$.*
3. *$\mathcal{A}$ sends its guess b'.*

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

We note that in the game above, $\mathcal{A}$ does not receive or have the ability to create a signature on m^*. For all signatures $\mathcal{A}$ can create by signing with $\mathsf{SK}_{\mathsf{abo}}$, $\mathsf{VK}_{\mathsf{abo}}$ and VK will behave identically.

Definition 21 (Splitting indistinguishability). *A splittable signature scheme § is said to be splitting indistinguishable if any PPT adversary $\mathcal{A}$ has negligible advantage in the following security game:*

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

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

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

In the game above, $\mathcal{A}$ is either given a system of $\sigma_{\mathsf{one}}, \mathsf{VK}_{\mathsf{one}}, \mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}}$ generated together by one call of $\mathsf{SetupSpl}$ or a "split" system of $(\sigma'_{\mathsf{one}}, \mathsf{VK}'_{\mathsf{one}}, \mathsf{SK}_{\mathsf{abo}}, \mathsf{VK}_{\mathsf{abo}})$ where the all but one keys are generated separately from the signature and key for the one message m^*. Since the correctness conditions do not link the behaviors for the all but one keys and the one message values, this split generation is not detectable by testing verification for the σ_{one} that $\mathcal{A}$ receives or for any signatures that $\mathcal{A}$ creates honestly by signing with $\mathsf{SK}_{\mathsf{abo}}$.

A.3 Security Properties of Iterators

Let $\mathsf{Itr} = (\mathsf{SetupItr}, \mathsf{ItrEnforce}, \mathsf{Iterate})$ be an interator with message space $\{0,1\}^\ell$ and state space $\S_\lambda$. We require the following notions of security.

Definition 22 (Indistinguishability of Setup). *An iterator Itr is said to satisfy indistinguishability of Setup phase if any PPT adversary $\mathcal{A}$'s advantage in the security game $\mathsf{Expt}_{\mathsf{Itr}}(1^\lambda, \mathcal{A})$ at most is negligible in λ, where $\mathsf{Expt}_{\mathsf{Itr}}$ is defined as follows.*

$\mathsf{Expt}_{\mathsf{Itr}}(1^\lambda, \mathcal{A})$

1. *The adversary $\mathcal{A}$ chooses a bound $T \in \Theta(2^\lambda)$ and sends it to challenger.*
2. *$\mathcal{A}$ sends k messages $m_1, \ldots, m_k \in \{0,1\}^\ell$ to the challenger.*
3. *The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{SetupItr}(1^\lambda, T)$. Else, it outputs $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{ItrEnforce}(1^\lambda, T, 1^k, \vec{m} = (m_1, \ldots, m_k))$.*
4. *$\mathcal{A}$ sends a bit b'.*

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

Definition 23 (Enforcing). *Consider any $\lambda \in \mathbb{N}$, $T \in \Theta(2^\lambda)$, $k < T$ and $m_1, \ldots, m_k \in \{0,1\}^\ell$. Let $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{ItrEnforce}(1^\lambda, T, \vec{m} = (m_1, \ldots, m_k))$ and $v_j = \mathsf{Iterate}^j(\mathsf{PP}_{\mathsf{Itr}}, v_0, (m_1, \ldots, m_j))$ for all $1 \leq j \leq k$. Then, Itr = ($\mathsf{SetupItr}$, $\mathsf{ItrEnforce}$, $\mathsf{Iterate}$) is said to be* enforcing *if*

$$v_k = \mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v', m') \implies (v', m') = (v_{k-1}, m_k).$$

Note that this is an information-theoretic property.