FE and iO for Turing Machines from Minimal Assumptions

Shweta Agrawal[1] and Monosij Maitra[2]

[1] IIT Madras, India
shweta.a@cse.iitm.ac.in
[2] IIT Madras, India
monosij@cse.iitm.ac.in

Abstract. We construct Indistinguishability Obfuscation (iO) and Functional Encryption (FE) schemes in the Turing machine model from the minimal assumption of compact FE for circuits (CktFE). Our constructions overcome the barrier of sub-exponential loss incurred by all prior work. Our contributions are:

1. We construct iO in the Turing machine model from the same assumptions as required in the circuit model, namely, sub-exponentially secure FE for circuits. The previous best constructions [41, 6] require sub-exponentially secure iO for circuits, which in turn requires sub-exponentially secure FE for circuits [5, 15].
2. We provide a new construction of single input FE for Turing machines with unbounded length inputs and optimal parameters from *polynomially* secure, compact FE for circuits. The previously best known construction by Ananth and Sahai [7] relies on iO for circuits, or equivalently, sub-exponentially secure FE for circuits.
3. We provide a new construction of multi-input FE for Turing machines. Our construction supports a fixed number of encryptors (say k), who may each encrypt a string $\mathbf{x}_i$ of *unbounded* length. We rely on sub-exponentially secure FE for circuits, while the only previous construction [10] relies on a strong knowledge type assumption, namely, public coin differing inputs obfuscation.

Our techniques are new and from first principles, and avoid usage of sophisticated iO specific machinery such as positional accumulators and splittable signatures that were used by all relevant prior work [41, 7, 6].

1 Introduction

The notion of indistinguishability obfuscation (iO) [11] seeks to garble programs such that the obfuscations of any two functionally equivalent programs are indistinguishable. While non-obvious at first what such a guarantee is good for, iO has emerged as a surprisingly powerful notion in cryptography, leading to many advanced cryptographic applications that were previously out of reach [27, 50, 23, 22, 12, 41, 14, 44, 26, 24, 46].

Functional encryption (FE) [49, 16, 48] is a generalization of public key encryption that enables fine grained access control on encrypted data. In FE, a secret key corresponds to a function f and ciphertexts correspond to strings from the domain of f. Given a function key SK_f and a ciphertext $\mathsf{CT}_\mathbf{x}$, the decryptor learns $f(\mathbf{x})$ and nothing else.

While an important primitive in its own right, FE has also been shown to imply iO, albeit with sub-exponential loss [5, 15]. Over the last few years, both primitives have received significant attention, with a rich body of work that attempts to support more general models of computation [12, 22, 41, 20, 25, 4, 21], rely on weaker assumptions [19, 30, 42, 13, 40, 8, 43, 45, 37, 38, 39], achieve stronger security [3, 19] and greater efficiency [6].

In this work, we make further progress towards the goal of basing iO and FE on minimal assumptions, in the Turing machine model of computation. This question has been studied extensively [34, 7, 12, 41, 22, 25, 20, 4, 21, 6] – we refer the reader to [7, 6] for a detailed discussion. Below, we summarize the state of art:

1. iO for Turing Machines with unbounded memory and bounded inputs are constructed in the works of Koppula et al. and Ananth et al. [41, 6]. Both works rely on the existence of sub-exponentially secure

iO for circuits along with other standard assumptions. We note that FE for circuits implies iO with sub-exponential loss, so when relying on FE for circuits, these works incur double sub-exponential loss.

2. For *single input* FE for Turing machines that accept unbounded length inputs and place no restriction on the description size or space complexity of the machine, the state of art is the work of Ananth and Sahai [7], which relies on the existence of iO for circuits.
3. For *multi-input* FE in the Turing machine model, the only known construction is [10], which relies on the existence of public coin differing inputs obfuscation (diO).

Our Results. We construct Indistinguishability Obfuscation (iO) and Functional Encryption (FE) schemes in the Turing machine model from the minimal assumption of compact FE for circuits (CktFE). Our constructions overcome the barrier of sub-exponential loss incurred by all prior work. Our contributions are:

1. We construct iO for Turing machines with bounded inputs and unbounded memory from the same assumptions as required by iO for circuits, namely, sub-exponentially secure FE for circuits. The previous best constructions [41, 6] require sub-exponentially secure iO for circuits, which in turn requires sub-exponentially secure FE for circuits [5, 15], resulting in double sub-exponential loss.
2. We provide a new construction of single input FE for Turing machines with unbounded inputs, achieving optimal parameters from *polynomially* secure, compact FE for circuits. The previously best known construction by Ananth and Sahai [7] relies on iO for circuits, or equivalently, sub-exponentially secure FE for circuits. We note that iO for circuits implies decomposable compact FE for circuits [27] (please see the full version [1]), so our construction also implies FE for TMs from iO for circuits.
3. We provide a new construction of multi-input FE for Turing machines. Our construction supports a fixed number of encryptors (say k), who may each encrypt a string $\mathbf{x}_i$ of *unbounded* length. We rely on sub-exponentially secure FE for circuits, while the only previous construction [10] relies on a strong knowledge type assumption, namely, public coin differing inputs obfuscation. The arity k supported by our scheme depends on the underlying multi-input CktFE scheme, for instance using [40], we can support $k = \mathsf{polylog}(\lambda)$.

Our constructions make use of FE for circuits that satisfy a mild property called *decomposablity*, which in turn can be constructed generically from FE for circuits (please see Appendix A). Decomposable FE, analogously to decomposable randomized encodings [9], roughly posits that a long string be encrypted bit by bit using shared randomness across bits. This property is already satisfied by all known constructions of CktFE in the literature to the best of our knowledge.

Our techniques are new and from first principles, and avoid usage of sophisticated iO specific machinery such as positional accumulators and splittable signatures that were used by all prior work [41, 7, 6]. Our work leverages the security notion of distributional indistinguishability (DI) for CktFE which was first considered by [31], who provided a construction for single input FE satisfying DI security assuming the existence of iO. We strengthen this result by constructing DI secure CktFE from standard CktFE. Please see Figure 1 for an overview of our results.

Additional Prior Work. Since iO is considered an inherently sub-exponential assumption and much stronger than the polynomial assumption of compact FE, replacing iO by FE in cryptographic constructions has already been studied extensively, for instance in the context of PPAD hardness [28], multi-input FE for circuits [19, 40] as well as trapdoor one-way permutations and universal samplers [29]. We note that aside from reliance on weaker, better understood assumptions, avoiding sub-exponential loss results in significantly more efficient schemes. We refer the reader to [29] for a detailed discussion.

Distributional indistinguishability was also considered in the context of output compressing randomized encodings [44]; indeed, this work implies that achieving DI security for FE for Turing machines with *long* outputs is impossible in the plain model. We note that our construction sidesteps this lower bound by considering Turing machines with a single output bit.

iO for TMs with unbounded memory has been constructed by [41, 6] as discussed above, other prior works were limited to bounded space constraints. We note that [6] additionally achieve constant overhead in the size of the obfuscated program as well as amortization, which we do not consider in this work. We also note that the work of [10] achieve miFE for TMs where the number of encrypting parties can be arbitrary, whereas we only support a-priori fixed, bounded number of parties.

The approach of using decomposable FE for circuits to construct FE for deterministic finite automata (DFA) in the *single key* setting was suggested by [2]. In this work we develop and significantly generalize their ideas. In particular, we handle the unbounded key setting in FE for TMs which necessitates dealing with the much more complex indistinguishability style definition, for which we develop new proof techniques which use a novel "sliding trapdoor" approach and leverage distributional indistinguishability. In contrast, since [2] use simulation security for single key FE, their proof must not contend with any of these challenges. Please see below for details.

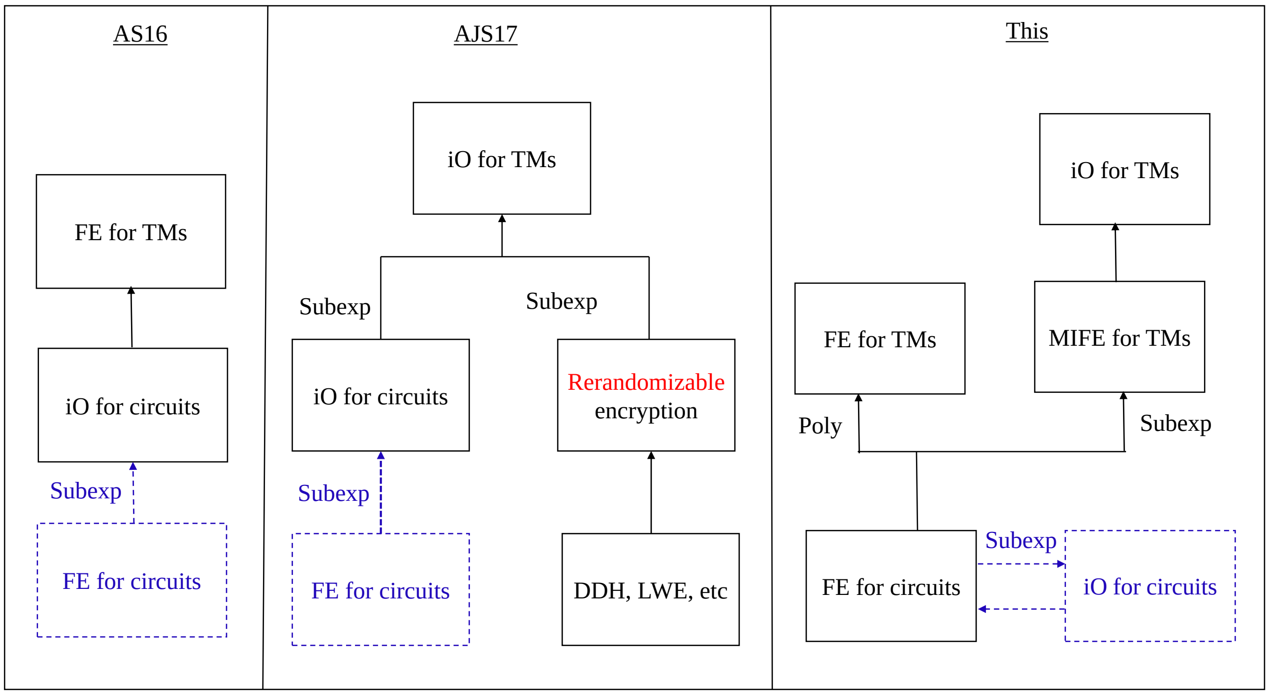

Fig. 1. Prior work and our results. The reductions with subexponential loss are specified, no specification implies standard polynomial loss. The dashed blue lines indicate primitives that are not actually used by the work in question; we add these to elucidate the relationship between primitives. We do not include [10] here since it relies on public coin diO.

Our Techniques. We describe an overview of our constructions, starting with single input FE, generalizing to multi-input FE and then building iO. All our constructions support the Turing machine model of computation. Our constructions rely on a single input FE scheme for *circuits*, denoted by CktFE, which satisfies *decomposability*. In Appendix A, we show that decomposable FE for circuits is implied by FE for circuits. Intuitively, decomposability means that the ciphertext $\mathsf{CT}_{\mathbf{x}}$ for a multi-bit message $\mathbf{x}$ be decomposable into multiple ciphertext components CT_i for $i \in |\mathbf{x}|$, one for each bit x_i of the message. Moreover, the ciphertext components encoding individual bits of a single input are tied together by common randomness, that is $\mathsf{CT}_i = \mathcal{E}(\mathsf{PK}, r, x_i)$ where $\mathcal{E}$ is an encoding function and r is common randomness used for all $i \in |\mathbf{x}|$[3]. The notion of decomposability has been widely studied and used in the context of randomized encodings, which may be seen as a special case of functional encryption; please see [9] as an example.

Single Input TMFE. Recall that a Turing machine at any time step reads a symbol, state pair and produces a new symbol which is written to the work tape, a new state and a left or right head movement. By assuming the Turing machine is *oblivious*, the head movements of the TM may be fixed; thus, at any given time step when a work tape cell is read, we can compute the next time step when the same work tape cell will be accessed. This reduces the output at any time step t to a symbol, state pair, where the state is read in the next time step $t + 1$ and the symbol is read at a future (fixed) time step $t' > t$.

Our construction uses two CktFE schemes, $1\mathsf{FE}_1$ and $1\mathsf{FE}_2$, where $1\mathsf{FE}_2$ is decomposable. Intuitively, $1\mathsf{FE}_1$ is used by the encryptor to encode the unbounded length input, while $1\mathsf{FE}_2$ is used to mimic the computation of the Turing machine, as we describe next. The ciphertext of $1\mathsf{FE}_2$ is divided into two parts, encoding input components (t, σ) and q respectively. Here, t is the current time step in the computation and σ, q are the current work-tape symbol and state respectively. We maintain the invariant that at any time step t in the computation, both components of the ciphertext have been computed using common randomness derived from $\mathsf{PRF}_\mathsf{K}((t\|\mathsf{salt}))$, where salt is an input chosen by the key generator and the PRF key K is chosen by the encryptor.

Now, to mimic the TM computation, we provide a function key for the Next functionality, that stores the transition table, receives as input the current (symbol, state) pair, computes the symbol to be written on the work tape and the next state using the transition table, derives the randomness using the PRF for the appropriate time step and outputs the encodings of the new (symbol, state) pair. In more detail, say the encryptor provides encodings of each input symbol x_i, for $i \in [|\mathbf{x}|]$, in addition to an encoding for the first (fixed) state $\mathsf{q_{st}}$, where the encodings of $(1, x_1)$ and $\mathsf{q_{st}}$ share the same randomness so that they may be concatenated to yield a complete ciphertext for $(1, x_1, \mathsf{q_{st}})$. Now, the function key may read input $(1, x_1, \mathsf{q_{st}})$, lookup the transition table and produce an encryption of the next state q_2 and the symbol to be written x'_2. The randomness used to encrypt q_2 is derived using a PRF as described above, and is the *same* as the randomness used by the encryptor to encode $(2, x_2)$. Hence, the two ciphertext components encoding $(2, x_2)$ and q_2 may be concatenated to yield a complete $1\mathsf{FE}_2$ ciphertext which may be again decrypted using the function key.

Now consider how to support writing on tape. Say the symbol x'_2 will be read at future fixed time step t'. Then the function key encodes the tuple (t', x'_2) using randomness $\mathsf{PRF}_\mathsf{K}((t'\|\mathsf{salt}))$. The state for time step t', say q' is computed at time step $t' - 1$, also using randomness $\mathsf{PRF}_\mathsf{K}((t'\|\mathsf{salt}))$. Thus, encodings of (t', x'_2) and q' may be joined together to yield a complete $1\mathsf{FE}_2$ ciphertext which may be decrypted to propagate the computation.

A detail brushed away by the above description is that the encryptor, given input $\mathbf{x}$, cannot compute randomness generated by a PRF which has input a value salt chosen by the key generator. This is handled by making use of an additional scheme $1\mathsf{FE}_1$, which re-encrypts ciphertexts provided by the encryptor via a ReRand functionality, using the requisite randomness. Note that we support inputs of unbounded length by leveraging the fact that CktFE schemes $1\mathsf{FE}_1, 1\mathsf{FE}_2$ support encryption of unbounded *number* of inputs, even if each must be of bounded length. Thus, the encryptor provides an unbounded number of $1\mathsf{FE}_1$ ciphertexts which are rerandomized and translated to ciphertexts under $1\mathsf{FE}_2$ using the ReRand function key provided by the key generator.

Encoding the PRF key. The above informal description hides an important detail – for the function key to produce ciphertext components using a PRF, it must have the key of the PRF, chosen by the encryptor[4], passed to it as input. Thus the ciphertext must additionally encode the PRF key along with

[3] Encoding of each bit may also use additional independent randomness, which is not relevant to the discussion here, and hence omitted.

[4] Note that the PRF key must be encoded in the ciphertext rather than function key since it is required to be hidden.

inputs (t, x, q). However, the ciphertext is constructed using randomness derived from the same PRF-resulting in circularity. We resolve this difficulty by using *constrained* PRFs [17, 36, 18], and having a ciphertext encode a PRF key that only allows computation of randomness for time steps *of the future*; this does not compromise its own security. For this constraint family, we provide a construction of cPRFs from standard assumptions. We believe this construction and the method of its application may be useful elsewhere[5].

More formally, our construction makes use of constrained, delegatable PRF for the function family $f_t : \{0,1\}^{2\cdot\lambda} \rightarrow \{0,1\}$ defined as follows.

$$\begin{aligned} f_t(x\|z) &= 1 \quad if \;\; x \geq t \\ &= 0 \quad otherwise \end{aligned}$$

We denote the constrained PRF key K_{f_t} by K_t for brevity. By the delegation property of constrained PRFs, we have that if $t' \geq t$ then $\mathsf{K}_{t'}$ can be derived from K_t. The proof requires the PRF to be punctured at a fixed point in each hybrid, we provide a construction of delegatable punctured PRF in the full version of the paper [1].

Proof Overview. While the above description of single input TMFE is natural and intuitive, the proof of indistinguishability based security is quite subtle and requires new techniques as we discuss next. For ease of exposition, we describe the proof overview for the case where the adversary makes a single key request corresponding to some TM M. We must argue that the challenge ciphertext, which is a sequence of $\mathsf{1FE}_1$ ciphertexts, together with ReRand and Next keys corresponding to a TM M, do not distinguish the bit b.

As discussed above, the $\mathsf{1FE}_1$ ciphertexts are decrypted using the ReRand key to produce a sequence of $\mathsf{1FE}_2$ ciphertexts, each corresponding to a time step in the TM execution (when the encoded symbol is read), which are in turn decrypted by Next keys to compute new $\mathsf{1FE}_2$ ciphertexts for future time steps. We may view the $\mathsf{1FE}_2$ ciphertexts as forming a chain, with each link of the chain corresponding to a single step of the TM computation, and each ciphertext producing (via decryption) a new ciphertext for the next time step, finally yielding the output when the TM halts (after T steps, say). Intuitively, since the output of the TM does not distinguish the bit b by admissibility of the TMFE adversary, we may argue by security of $\mathsf{1FE}_2$ that the ciphertext at the penultimate step $T-1$ also does not distinguish b, which implies that the ciphertext at step $T-2$ hides b and so on, ultimately yielding indistinguishability of the entire chain, and hence of the $\mathsf{1FE}_1$ challenge ciphertext.

Formalizing this intuitive argument is quite tricky. A natural approach would be to consider a sequence of hybrids, one corresponding to each link in the chain, and switch the $\mathsf{1FE}_2$ ciphertexts one by one starting from the end of the chain. While intuitive, this idea is misleading – note that a naive implementation of this idea would lead to a chain which is "broken": namely, its first links correspond to $b = 0$, and last links to $b = 1$. Since the ciphertext at a given step is decrypted to compute the ciphertext at the next step, a ciphertext corresponding to $b = 0$ cannot in general output a ciphertext for $b = 1$.

A standard approach to deal with this difficulty is to embed a "trapdoor" mode within the functionality [3, 5, 19] which lets us "hardwire" the ciphertexts that must be output by decryption directly in the key, allowing decryption to yield an inconsistent chain. However, this approach also fails in our case, since the length of the chain is unbounded and there isn't sufficient space in the key to incorporate all its values.

Our Approach: "Sliding" Trapdoors. We deal with this difficulty by designing a novel "sliding-window" trapdoor approach which lets us hardwire the decryption chain "piece by piece". In more detail, we start with the last two time steps $(T, T-1)$, program the key to produce the output corresponding to $b = 1$ for time step T and $b = 0$ for $T-1$, then transition to a world where the output corresponds to $b = 1$

[5] For instance, a similar situation w.r.t circularity arises in the original garbled RAM construction of Lu and Ostrovsky [47].

for both T and $T-1$. At this point, the hardwiring of the output for time step T is redundant, since the ciphertext output by the decryption process at time step $T-1$ automatically computes the output coresponding to $b=1$ at time step T. Thus, we may now *slide* the trapdoor to program to the next pair $(T-1, T-2)$, switching the decryption output at time step $T-2$ to $b=1$ and so on, until the entire chain corresponds to $b=1$.

Intuitively, we are "programming" the decryption only for outputs at both ends of the "broken link", so that preceding links are generated using $b=0$ and subsequent links are generated using $b=1$. We leverage the fact that the chain links corresponding to future time-steps are encoded *implicitly* in a given time step – hence if we manage to hide the chain inconsistency at a certain position i, this implies that the remainder of the chain is constructed using the bit encoded at step i. Formalizing this argument requires a great deal of care, as we must keep track of the "target" time steps corresponding to the two ends of the broken link that are being programmed, the time steps at which the symbol and state ciphertexts are generated to be "consumed" at the target time-steps, the particular values that must be encoded in the symbol, state fields in both cases as well as the key that is being handled at a given time in the proof. For more details, please see Section 3.3.

Generalising to Multi-Input FE for Turing machines. For the k party setting, a natural idea is to have each party encrypt its own input $\mathbf{x}_i$, and use a k input CktFE scheme kFE [19, 40], to "aggregate" these into the "input" ciphertext $\mathsf{CT}(\mathbf{x})$ for one long input $\mathbf{x} = (\mathbf{x}_1 \| \mathbf{x}_2 \| \ldots \| \mathbf{x}_k)$, under a different CktFE scheme $\mathsf{1FE}$. Note that the length of $\mathbf{x}$ is unknown hence it may not be encoded "all at once" but must be encoded bit by bit as in the previous scheme. Now, by additionally providing the $\mathsf{1FE}$ ciphertext encoding the start state of the Turing machine $\mathsf{CT}(\mathsf{q_{st}})$, and a function key to compute the transition table of the TM as in the previous scheme, we may proceed with the computation exactly as before.

Formalizing this idea must contend with several hurdles. In the multi-input setting, the i^{th} encryptor may encode multiple inputs and functionality permits "mix and match" of ciphertexts in the sense that any input encoded by party i may be combined with any input encoded by parties $j \in [k],\ j \neq i$. Therefore, if each of k parties encodes T ciphertexts, there are T^k valid input combinations that the TM may execute on. However, when the TM is executing on any input combination, we must ensure that it cannot mix and match symbol, state pairs across different input combinations. Moreover, an encryption for a symbol, state pair produced by some machine M_i should not be decryptable by any machine M_j for $j \neq i$. These issues are handled by careful design of the aggregate functionality to ensure that an execution thread of any input combination by any machine is separate from any other. The proof extends naturally from the single input case. Please see Section 4 for details.

Distributional Indistinguishability. As discussed above, our constructions rely on the security notion of *distributional indistinguishability* (DI) for functional encryption for circuits [31]. Intuitively, this notion says that if the outputs produced by a circuit on two input distributions are merely indistinguishable (as against exactly equal), then the ciphertexts encoding those inputs must also be indistinguishable. In the full version [1] we give a construction of DI secure single input FE from standard FE.

Indistinguishability Obfuscation. Constructing iO for TMs given miFE for TM is straightforward, and adapts the miFE to iO circuit compiler by [33] to the TM setting. As in the circuit case, an miFE for TM that supports two ciphertext queries and single key query suffices for this transformation. Please see Section 5 for details. Since our security proof for miFE for TM is tight, this compiler yields iO for TM from sub-exponentially secure FE for circuits rather than sub-exponentially secure iO for circuits.

Organization of the paper. Definitions and preliminaries are provided in Section 2 as well as the full version [1]. In Section 3, we provide our construction for single input FE for Turing machines. In Section 4, we provide our construction for multi-input FE for Turing machines for any fixed arity k and in Section 5 we describe our iO for TMs for bounded inputs.

2 Preliminaries

In this section, we define some notation and preliminaries that we require.

Notation. We begin by defining the notation that we will use throughout the paper. We use bold letters to denote vectors and the notation $[a, b]$ to denote the set of integers $\{k \in \mathbb{N} \mid a \leq k \leq b\}$. We use $[n]$ to denote the set $[1, n]$. Concatenation is denoted by the symbol $\|$.

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\text{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\text{poly}(n)$ to denote a polynomial function of n. We use the abbreviation PPT for probabilistic polynomial-time. We say an event occurs with *overwhelming probability* if its probability is $1 - \text{negl}(n)$. The function $\log x$ is the base 2 logarithm of x.

2.1 Definitions: FE for Circuits

In this section, we define functional encryption for circuits, in both the single and multi-input setting.

Single Input Functional Encryption for Circuits Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ denote ensembles where each $\mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ is a finite set. Let $\mathcal{F} = \left\{\mathcal{F}_\lambda\right\}_{\lambda \in \mathbb{N}}$ denote an ensemble where each $\mathcal{F}_\lambda$ is a finite collection of circuits, and each circuit $f \in \mathcal{F}_\lambda$ takes as input a string $\mathbf{x} \in \mathcal{X}_\lambda$ and outputs $f(\mathbf{x}) \in \mathcal{Y}_\lambda$.

A functional encryption scheme CktFE for $\mathcal{F}$ consists of four algorithms
$\mathsf{CktFE} = (\mathsf{CktFE.Setup}, \mathsf{CktFE.Keygen}, \mathsf{CktFE.Enc}, \mathsf{CktFE.Dec})$ defined as follows.

- $\mathsf{CktFE.Setup}(1^\lambda)$ is a PPT algorithm that takes as input the unary representation of the security parameter and outputs the master public and secret keys $(\mathsf{PK}, \mathsf{MSK})$. Sometimes, the $\mathsf{CktFE.Setup}$ algorithm may also accept as input a parameter 1^ℓ, denoting the length of the input. In this case, the input lives in domain $\mathcal{X}^\ell$.
- $\mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$ is a PPT algorithm that takes as input the master secret key MSK and a circuit $f \in \mathcal{F}_\lambda$ and outputs a corresponding secret key SK_f.
- $\mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x})$ is a PPT algorithm that takes as input the master public key PK and an input message $\mathbf{x} \in \mathcal{X}_\lambda$ and outputs a ciphertext CT.
- $\mathsf{CktFE.Dec}(\mathsf{SK}_f, \mathsf{CT}_\mathbf{x})$ is an (a deterministic) algorithm that takes as input the secret key SK_f and a ciphertext $\mathsf{CT}_\mathbf{x}$ and outputs $f(\mathbf{x})$.

Definition 1 (Correctness). *A functional encryption scheme* CktFE *is correct if for all* $\lambda \in \mathbb{N}$*, all* $f \in \mathcal{F}_\lambda$ *and all* $x \in \mathcal{X}_\lambda$*,*

$$\Pr\left[\begin{array}{l}(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{CktFE.Setup}(1^\lambda); \\ \mathsf{CktFE.Dec}\Big(\mathsf{CktFE.Keygen}(\mathsf{MSK}, f), \mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x})\Big) \neq f(\mathbf{x})\end{array}\right] = \text{negl}(\lambda)$$

where the probability is taken over the coins of $\mathsf{CktFE.Setup}$, $\mathsf{CktFE.Keygen}$*, and* $\mathsf{CktFE.Enc}$.

Definition 2 (Compactness [5]). *A functional encryption scheme for circuits is said to be compact if for any input message* $\mathbf{x}$*, the running time of the encryption algorithm is polynomial in the security parameter and the size of* $\mathbf{x}$*. In particular, it does not depend on the circuit description size or the output length of any function* f *supported by the scheme.*

A weaker version of compactness, known as **succinct** or semi-compact FE, allows the run time of the encryption algorithm to depend on the output length of the functions. Equivalently, a semi-compact FE scheme is simply a compact FE scheme when we restrict our attention to functions with single-bit outputs.

Distributional Indistinguishability for Circuit FE. In this section we define the notion of distributional indistinguishability for functional encryption for circuits. The notion was first defined by [31, Sec 3.4] in the context of reusable garbled circuits, i.e. single key functional encryption but may be generalized to the multi-key setting in a straightforward way. Intuitively, this notion says that if the outputs produced by a circuit on two input distributions are indistinguishable, then the ciphertexts encoding those inputs must also be indistinguishable.

Definition 3. *A functional encryption scheme $\mathcal{F}$ for a circuit family $\mathcal{G}$ is secure in the distributional indistinguishability game, if for all* PPT *adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter λ:*

1. *Public Key: Challenger returns* PK *to the adversary.*
2. *Pre-Challenge Key Queries: $\mathcal{A}$ may adaptively request keys for any circuits $g_i \in \mathcal{G}$. In response, $\mathcal{A}$ is given the corresponding keys SK_{g_i}. This step may be repeated any polynomial number of times by the attacker.*
3. *Challenge Declaration: $\mathcal{A}(1^\lambda, \mathsf{PK})$ outputs two ensembles of challenge distributions $\big(D_0(\lambda), D_1(\lambda)\big)$[6] to the challenger, subject to the restriction that for any $\mathbf{x}_0 \leftarrow D_0, \mathbf{x}_1 \leftarrow D_1$, it holds that $g_i(\mathbf{x}_0) \overset{c}{\approx} g_i(\mathbf{x}_1)$ for all i.*
4. *Challenge CT: $\mathcal{A}$ requests the challenge ciphertext, to which challenger chooses a random bit b, samples $\mathbf{x}_b \leftarrow D_b$ and returns the ciphertext $\mathsf{CT}_{\mathbf{x}_b}$.*
5. *Key Queries: The adversary may continue to request keys for additional functions g_i, subject to the same restriction that for any $\mathbf{x}_0 \leftarrow D_0, \mathbf{x}_1 \leftarrow D_1$, it holds that $g_i(\mathbf{x}_0) \overset{c}{\approx} g_i(\mathbf{x}_1)$ for all i.*
6. *$\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.*

The advantage *of $\mathcal{A}$ is the absolute value of the difference between its success probability and $1/2$. In the selective game, the adversary is required to declare the challenge distributions in the very first step, without seeing the public key.*

Comparison with Standard Indistinguishability. We note that the standard insitinguishability game is implied by the above by restricting the adversary to choose distributions D_0, D_1 above to simply be two messages $\mathbf{x}_0, \mathbf{x}_1$ with probability 1 and requesting keys that satisfy $g_i(\mathbf{x}_0) = g_i(\mathbf{x}_1)$ for all i, which is a special case of $g_i(\mathbf{x}_0) \overset{c}{\approx} g_i(\mathbf{x}_1)$.

Decomposable functional encryption for circuits. In this section, we recall the notion of *decomposable functional encryption* (DFE) defined by [2]. Decomposable functional encryption is analogous to the notion of decomposable randomized encodings [9]. Intuitively, decomposability requires that the public key PK and the ciphertext $\mathsf{CT}_\mathbf{x}$ of a functional encryption scheme be decomposable into components PK_i and CT_i for $i \in [|\mathbf{x}|]$, where CT_i depends on a single deterministic bit x_i and the public key component PK_i. In addition, the ciphertext may contain components that are independent of the message and depend only on the randomness.

Formally, let $\mathbf{x} \in \{0,1\}^k$. A functional encryption scheme is said to be decomposable if there exists a deterministic function $\mathcal{E} : \mathcal{P} \times \{0,1\} \times \mathcal{R}_1 \times \mathcal{R}_2 \rightarrow \mathcal{C}$ such that:

1. The public key may be interpreted as $\mathsf{PK} = (\mathsf{PK}_1, \ldots, \mathsf{PK}_k, \mathsf{PK}_{\mathsf{indpt}})$ where $\mathsf{PK}_i \in \mathcal{P}$ for $i \in [k]$. The component $\mathsf{PK}_{\mathsf{indpt}} \in \mathcal{P}^j$ for some $j \in \mathbb{N}$.
2. The ciphertext may be interpreted as $\mathsf{CT}_\mathbf{x} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathsf{CT}_{\mathsf{indpt}})$, where

$$\mathsf{CT}_i = \mathcal{E}\left(\mathsf{PK}_i, x_i, r, \hat{r}_i\right) \forall i \in [k] \quad \text{and} \quad \mathsf{CT}_{\mathsf{indpt}} = \mathcal{E}\left(\mathsf{PK}_{\mathsf{indpt}}, r, \hat{r}\right)$$

   Here $r \in \mathcal{R}_1$ is common randomness used by all components of the encryption. Apart from the common randomness r, each CT_i may additionally make use of independent randomness $\hat{r}_i \in \mathcal{R}_2$.

[6] We omit the parameter λ in what follows for brevity of notation.

We note that if a scheme is decomposable "bit by bit", i.e. into k components for inputs of size k, it is also decomposable into components corresponding to any partition of the interval $[k]$. Thus, we may decompose the public key and ciphertext into any $i \leq k$ components of length k_i each, such that $\sum k_i = k$. We will sometimes use $\bar{\mathcal{E}}(\mathbf{y})$ to denote the tuple of function values obtained by applying $\mathcal{E}$ to each component of a vector, i.e. $\bar{\mathcal{E}}(\mathsf{PK}, \mathbf{y}, r) \triangleq \Big(\mathcal{E}(\mathsf{PK}_1, y_1, r, \hat{r}_1), \ldots, \mathcal{E}(\mathsf{PK}_k, y_k, r, \hat{r}_k)\Big)$, where $|\mathbf{y}| = k$. We assume that given the security parameter, the spaces $\mathcal{P},\ \mathcal{R}_1,\ \mathcal{R}_2,\ \mathcal{C}$ are fixed, and the length of the message $|\mathbf{x}|$ can be any polynomial.

Multi-Input Functional Encryption for Circuits. We define the notion of private-key t-input functional encryption for circuits here. Our definition follows that of [40].

Let $\forall i \in [t], \mathcal{X}_i = \{(\mathcal{X}_i)\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ be ensembles of finite sets, and let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be an ensemble of finite t-ary function families. For each $\lambda \in \mathbb{N}$, each function $f \in \mathcal{F}_\lambda$ takes as input t strings, $\mathbf{x}_1 \in (\mathcal{X}_1)_\lambda, \ldots, \mathbf{x}_t \in (\mathcal{X}_t)_\lambda$, and outputs a value $f(\mathbf{x}_1, \ldots, \mathbf{x}_t) \in \mathcal{Y}_\lambda$.

A private-key t-input functional encryption scheme t-CktFE for $\mathcal{F}$ consists of four algorithms t-$\mathsf{CktFE} = (t\text{-}\mathsf{CktFE.Setup}, t\text{-}\mathsf{CktFE.Keygen}, t\text{-}\mathsf{CktFE.Enc}, t\text{-}\mathsf{CktFE.Dec})$ defined as follows.

- t-$\mathsf{CktFE.Setup}(1^\lambda)$ is a PPT algorithm that takes as input the unary representation of the security parameter and outputs the master secret key MSK.
- t-$\mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$ is a PPT algorithm that takes as input the master secret key MSK and a circuit $f \in \mathcal{F}_\lambda$ and outputs a corresponding secret key SK_f.
- t-$\mathsf{CktFE.Enc}(\mathsf{MSK}, \mathbf{m}, \mathsf{ind})$ is a PPT algorithm that takes as input the master secret key MSK, an input message $\mathbf{m} = \mathbf{x}_i \in (\mathcal{X}_i)_\lambda$ if $\mathsf{ind} = i, i \in [t]$, and outputs a ciphertext $\mathsf{CT}_{\mathsf{ind}}$.
- t-$\mathsf{CktFE.Dec}(\mathsf{SK}_f, (\mathsf{CT}_1, \ldots, \mathsf{CT}_t))$ is an (a deterministic) algorithm that takes as input the secret key SK_f and t ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_t$ and outputs a string $y \in \mathcal{Y}_\lambda \cup \perp$.

Definition 4 (Correctness). *A private-key t-input functional encryption scheme t-CktFE is correct if for all $\lambda \in \mathbb{N}$, $f \in \mathcal{F}_\lambda$ and all $(\mathbf{x}_1, \ldots, \mathbf{x}_t) \in (\mathcal{X}_1)_\lambda \times \ldots \times (\mathcal{X}_t)_\lambda$,*

$$\Pr\left[\begin{array}{c} t\text{-}\mathsf{CktFE.Dec}\Big(t\text{-}\mathsf{CktFE.Keygen}(\mathsf{MSK}, f), \big(t\text{-}\mathsf{CktFE.Enc}(\mathsf{MSK}, \mathbf{x}_1, 1), \ldots, \\ t\text{-}\mathsf{CktFE.Enc}(\mathsf{MSK}, \mathbf{x}_t, t)\big)\Big) \neq f(\mathbf{x}_1, \ldots, \mathbf{x}_t) \end{array}\right] = \mathrm{negl}(\lambda)$$

Here, $\mathsf{MSK} \leftarrow t\text{-}\mathsf{CktFE.Setup}(1^\lambda)$ *and probability is taken over the random coins of* t-$\mathsf{CktFE.Setup}$, t-$\mathsf{CktFE.Enc}$ *and* t-$\mathsf{CktFE.Keygen}$.

Distributional Indistinguishability. We define the notion of distributional indistinguishability for a t-input functional encryption scheme for circuits. To begin, we describe a valid t-input adversary.

Definition 5 (Valid t-Input Adversary). *A* PPT *algorithm $\mathcal{A}$ is a valid t-input adversary if for all private-key t-input functional encryption schemes over message space $(\mathcal{X}_1)_\lambda \times \ldots \times (\mathcal{X}_t)_\lambda$, and a circuit space $\mathcal{F}$, for any (f_0, f_1) queried by the adversary, and any t pairs of input distribution ensembles $(D_{01}(\lambda), D_{11}(\lambda)), \ldots, (D_{0t}(\lambda), D_{1t}(\lambda))$[7] output by the adversary such that D_{bj} is a distribution over $\mathcal{X}_j$ for $b \in \{0,1\}$, $j \in [t]$, it holds that*

$$f_0(\mathbf{x}_{01}, \ldots, \mathbf{x}_{0t}) \overset{c}{\approx} f_1(\mathbf{x}_{11}, \ldots, \mathbf{x}_{1t}),$$

where $\mathbf{x}_{bj} \leftarrow D_{bj}$ for $b \in \{0,1\}$, $j \in [t]$.

We define the following game between a challenger and an adversary:

[7] We omit the argument λ where it is implicit for notational brevity.

1. **Key Queries.** $\mathcal{A}$ may adaptively submit key requests for pairs of functions $(f_0, f_1) \in \mathcal{F}$. In response, $\mathcal{A}$ is given the corresponding keys SK_{f_b} for some random bit b chosen by the challenger. This step may be repeated any polynomial number of times by the attacker.
2. **Ciphertext Queries.** $\mathcal{A}(1^\lambda)$ submits ciphertext requests for pairs of challenge distribution ensembles $(D_{01}, D_{11}), \ldots, (D_{0t}, D_{1t})$ to the challenger. The challenger samples $\mathbf{x}_j \leftarrow D_{bj}$ for $j \in [t]$ and returns t-$\mathsf{CktFE.Enc}(\mathsf{MSK}, \mathbf{x}_j, j), \forall j \in [t]$. This step may be repeated any polynomial number of times by the attacker.
3. **Guess.** $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

In the above definition, ciphertext and key queries may be interspersed in any order. The *advantage* of $\mathcal{A}$ is the absolute value of the difference between its success probability and $1/2$. In the *selective* game, the adversary is required to declare the challenge ciphertext distributions in the very first step, without seeing the public key.

Definition 6. *A t-input functional encryption scheme t-CktFE for a circuit family $\mathcal{F}$ is secure in the distributional indistinguishability game, if for all valid* PPT *adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.*

We note that the standard indistinguishability game is the special case where the adversary submits challenge messages rather than distributions and all queried functions must output exactly the same rather than indistinguishable values.

2.2 Definitions: FE for Turing Machines

In this section, we will define functional encryption for Turing Machines (TM). We denote the runtime (i.e. number of steps the head takes) by $\mathsf{runtime}(M, \mathbf{w})$.

Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of Turing machines with alphabet $\Sigma = \{\Sigma_\lambda\}_{\lambda \in \mathbb{N}}$ and the running time upper-bounded by a polynomial in λ. A functional encryption scheme TMFE for a Turing machine family $\mathcal{M}$ consists of four algorithms $\mathsf{TMFE} = (\mathsf{TMFE.Setup}, \mathsf{TMFE.KeyGen}, \mathsf{TMFE.Enc}, \mathsf{TMFE.Dec})$ defined as follows.

– $\mathsf{TMFE.Setup}(1^\lambda)$ is a PPT algorithm that takes as input the unary representation of the security parameter and outputs the master public and secret keys $(\mathsf{PK}, \mathsf{MSK})$.
– $\mathsf{TMFE.KeyGen}(\mathsf{MSK}, M)$ is a PPT algorithm that takes as input the master secret key MSK and a TM M and outputs a corresponding secret key SK_M.
– $\mathsf{TMFE.Enc}(\mathsf{PK}, \mathbf{x})$ is a PPT algorithm that takes as input the master public key PK, and an input message $\mathbf{x} \in \Sigma_\lambda^*$ of arbitrary length, outputs a ciphertext $\mathsf{CT}_\mathbf{x}$.
– $\mathsf{TMFE.Dec}(\mathsf{SK}_M, \mathsf{CT}_\mathbf{x})$ is an (a deterministic) algorithm that takes as input the secret key SK_M and a ciphertext $\mathsf{CT}_\mathbf{x}$ and outputs a bit b.

Correctness is defined analogously to the circuit setting.

Efficiency [7]. The efficiency property of a public-key FE scheme for Turing machines says that the algorithm $\mathsf{TMFE.Setup}$ on input 1^λ should run in time polynomial in λ, $\mathsf{TMFE.KeyGen}$ on input the Turing machine M and the master key MSK should run in time polynomial in $(\lambda, |M|)$, $\mathsf{TMFE.Enc}$ on input a message $\mathbf{x}$ and the public key should run in time polynomial in $(\lambda, |\mathbf{x}|)$. Finally, $\mathsf{TMFE.Dec}$ on input a functional key of M and an encryption of $\mathbf{x}$ should run in time polynomial in $(\lambda, |M|, |\mathbf{x}|, \mathsf{runtime}(M, \mathbf{x}))$.

The multi-input case may be defined as in the circuit setting.

Indistinguishability Obfuscation for Turing Machines As in prior work, we construct iO for Turing machines (TMs) in the setting where the input length is fixed a-priori. A uniform P.P.T machine iO is an indistinguishability obfuscator for a class of Turing machines $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ with input length L, if the following conditions are satisfied:

1. **Correctness.** For all security parameters $\lambda \in \mathbb{N}$, for any $M \in \mathcal{M}_\lambda$ and every input $\mathbf{x} \in \{0,1\}^{\leq L}$, we have that:
$$\Pr\left[M' \leftarrow \mathsf{iO}(1^\lambda, M, L) : M'(\mathbf{x}) = M(\mathbf{x})\right] = 1$$
where the probability is taken over the coin-tosses of the obfuscator iO.
2. **Indistinguishability of Equivalent TMs.** For every ensemble of pairs of Turing machines $\{M_{0,\lambda}, M_{1,\lambda}\}_{\lambda\in\mathbb{N}}$, such that $M_{0,\lambda}(\mathbf{x}) = M_{1,\lambda}(\mathbf{x})$ for every $\mathbf{x} \in \{0,1\}^{\leq L}$ and $\mathsf{runtime}(M_{0,\lambda}, \mathbf{x}) = \mathsf{runtime}(M_{1,\lambda}, \mathbf{x})$, we have that the following ensembles of pairs of distributions are indistinguishable to any PPT Adv:
$$\left\{M_{0,\lambda}, M_{1,\lambda}, \mathsf{iO}(1^\lambda, M_{0,\lambda})\right\} \overset{c}{\approx} \left\{M_{0,\lambda}, M_{1,\lambda}, \mathsf{iO}(1^\lambda, M_{1,\lambda})\right\}$$
3. **Succinctness.** For all security parameters $\lambda \in \mathbb{N}$, for any $M \in \mathcal{M}_\lambda$, we have that the running time of $\mathsf{iO}(1^\lambda, M, L)$ is $\text{poly}(\lambda, |M|, L)$ and the evaluation time of $\mathsf{iO}(M)$ on input $\mathbf{x}$ where $\mathbf{x} \in \{0,1\}^{\leq L}$, is $\text{poly}(|M|, L, t)$ where $t = \mathsf{runtime}(M, \mathbf{x})$.

2.3 Constrained Pseudorandom Functions

Constrained pseudorandom functions (introduced concurrently by Boneh and Waters (CCS 2013), Boyle, Goldwasser, and Ivan (PKC 2014), and Kiayias, Papadopoulos, Triandopoulos, and Zacharias (CCS 2013)), are pseudorandom functions (PRFs) that allow the owner of the secret key K to compute a constrained key K_f , such that anyone who possesses K_f can compute the output of the PRF on any input x such that $f(x) = 1$ for some predicate f. The security requirement of constrained PRFs state that the PRF output must still look indistinguishable from random for any x such that $f(x) = 0$. We will also require the property of delegatability, formalized below.

Definition 7. *[17] Let $F : \{0,1\}^{\mathsf{seed}(\lambda)} \times \{0,1\}^{\mathsf{in}(\lambda)} \to \{0,1\}^{\mathsf{out}(\lambda)}$ be an efficient function, where* seed, in *and* out *are all polynomials in the security parameter λ. We say that F is a delegatable constrained pseudorandom function with respect to a set system $\mathcal{S} \subseteq 2^{\{0,1\}^{\mathsf{in}(\lambda)}}$ if there exist algorithms* $(\mathsf{Setup}, \mathsf{Constrain}, \mathsf{Eval}, \mathsf{KeyDel})$ *that satisfy the following:*

- $\mathsf{Setup}(1^\lambda, 1^{\mathsf{in}(\lambda)})$ *outputs a pair of keys* pk, sk.
- $\mathsf{Constrain}(\mathsf{sk}, S)$ *outputs a constrained key K_S which enables evaluation of $F(\mathsf{sk}, \mathbf{x})$ on all $\mathbf{x} \in S$ and no other* $\mathbf{x}$.
- $\mathsf{KeyDel}(K_S, S')$ *outputs a constrained key $K_{S\cap S'}$ which enables the evaluation of $F(\mathsf{sk}, \mathbf{x})$ for all $\mathbf{x} \in S \cap S'$ and no other $\mathbf{x}$. We note that in systems where* KeyDel *is supported, the* $\mathsf{Constrain}$ *algorithm above can be expressed as a special case of* KeyDel *by letting* sk *correspond to the set of all inputs, i.e.* $\mathsf{sk} = K_{\{0,1\}^{\mathsf{in}(\lambda)}}$.
- $\mathsf{Eval}(K_S, \mathbf{x})$ *outputs $F(\mathsf{sk}, \mathbf{x})$ if $\mathbf{x} \in S$, $\perp$ otherwise.*

Please refer to the full version [1] for the definition of security.

3 Construction: Single Input FE for Turing Machines

In this section, we construct a single input functional encryption scheme for Turing machines, denoted by TMFE from the following ingredients:

1. Two compact functional encryption schemes for circuits, $\mathsf{1FE}_1$ and $\mathsf{1FE}_2$. We will assume that the scheme $\mathsf{1FE}_2$ is decomposable as defined in the preliminaries.
2. A symmetric encryption scheme $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$.
3. A delegatable constrained pseudorandom function (cPRF), denoted by F which supports T delegations for the function family $f_t : \{0,1\}^{2\cdot\lambda} \to \{0,1\}$ defined as follows. Let x, t denote integers whose binary representations are $\mathbf{x}, \mathbf{t}$ of λ bits. Then,

$$f_t(\mathbf{x}\|\mathbf{z}) = 1, \text{if } \ x \geq t \text{ and } 0 \text{ otherwise}$$

Intuitively, the function is parametrized by a value t and evaluates to 1 if the first half of its input, $x \geq t$. We will denote the constrained PRF key K_{f_t} corresponding to function f_t by K_t for ease of notation. By the delegation property of constrained PRFs, we have that if $t' \geq t$ then $\mathsf{K}_{t'}$ can be derived from K_t. In our construction the parameter t will represent the time step in the computation, which means that a PRF key of the current time step can be used to derive PRF keys for future time steps. We will denote a PRF for this functionality by F. The security proof makes use a punctured version of the above cPRF, please see the full version [1] for details.

3.1 Construction

Below we provide our construction for single input FE for Turing machines.

Notation. Note that since $\mathsf{1FE}_2$ is decomposable, there exists an encoding function $\mathcal{E}$ which encodes each bit of the input and since it is compact, the output length of $\mathcal{E}$ is independent of the circuit class supported by $\mathsf{1FE}_2$. Thus, by choosing the encoding function first, the CktFE scheme may support a circuit class that outputs its own ciphertext components. We denote by $\bar{\mathcal{E}}$ the encoding function $\mathcal{E}$ applied bitwise to a vector, i.e. $\bar{\mathcal{E}}(\mathbf{w}) = \mathcal{E}(w_1) \dots \mathcal{E}(w_n)$.

$\mathsf{TMFE.Setup}(1^\lambda)$: Upon input the security parameter 1^λ, do the following:

1. Let $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda)$, where $\mathsf{1FE}_2$ is a decomposable functional encryption scheme for the circuit family

$$\mathsf{Next} : \Big(\big(\{\mathsf{SYM}\} \times \{0,1\}^{4\lambda} \times \Sigma \times \mathsf{Trap} \big) \times \big(\{\mathsf{ST}\} \times \mathcal{Q} \big) \Big) \to \big(\mathcal{C}^{\mathsf{1FE}_2} \big)^2 \cup \{\mathsf{ACC}, \mathsf{REJ}, \perp\}$$

Here, Σ and $\mathcal{Q}$ are the alphabet and state space respectively of the Turing machine family. The tokens SYM and ST are flags denoting a symbol and a state respectively. The set $\{0,1\}^{4\lambda}$ encodes in order, a random value key-id associated with a TM M, a cPRF key, the current time step in the computation and the length of the input string, each of λ bits. Here, Trap is a data structure of fixed polynomial length which will be used in the proof. Since we do not need it in the construction, we do not discuss it here, please see Figure 6 for its definition. $\mathcal{C}^{\mathsf{1FE}_2}$ denotes the ciphertext space of $\mathsf{1FE}_2$, and ACC and REJ are bits indicating accepting and a rejecting states of a TM respectively.

2. Let $(\mathsf{1FE}_1.\mathsf{PK}, \mathsf{1FE}_1.\mathsf{MSK}) \leftarrow \mathsf{1FE}_1.\mathsf{Setup}(1^\lambda)$, where $\mathsf{1FE}_1$ is a compact, public-key CktFE scheme for the circuit family

$$\mathsf{ReRand} : \big(\{0,1\}^{3\lambda} \times \Sigma \times \mathsf{Trap} \big) \to \mathcal{C}^{\mathsf{1FE}_2} \times \big(\mathcal{C}^{\mathsf{1FE}_2} \cup \{\perp\} \big)$$

Again, $\{0,1\}^{3\lambda}$ encodes in order, a root cPRF key, a time step and the length of the input string respectively, while Σ, Trap and $\mathcal{C}^{\mathsf{1FE}_2}$ are as described above.

3. Output $\mathsf{PK} = \mathsf{1FE}_1.\mathsf{PK}$ and $\mathsf{MSK} = (\mathsf{1FE}_1.\mathsf{MSK}, (\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}))$.

$\mathsf{TMFE.Enc}(\mathsf{PK}, \mathbf{w})$: Upon input the public key PK, and message $\mathbf{w}$ of arbitrary length $\ell = |\mathbf{w}|$, do the following:

1. Sample the root key K_0 for function f_t where $t = 0$ for the cPRF F described above.

2. For $i \in [\ell]$, let $\mathsf{CT}_i = 1\mathsf{FE}_1.\mathsf{Enc}(\mathsf{PK}, (\mathsf{K}_0, i, \ell, w_i, \mathsf{Trap}))$, where Trap is a data structure which is only relevant in the proof. Here, all fields of Trap are set to $\perp$ except a flag $\mathsf{Trap.mode\text{-}real} = 1$ which indicates that we are in the real world. Please see Figure 6 for the definition of Trap.
3. Output $\mathsf{CT}_{\mathbf{w}} = \{\mathsf{CT}_i\}_{i \in [\ell]}$.

$\mathsf{TMFE.KeyGen}(\mathsf{MSK}, M)$: Upon input the master secret key MSK and the description of a Turing machine M, do the following. We will assume, w.l.o.g. that the TM is oblivious (see [1] for a justification) and $\mathsf{q_{st}} \in \mathcal{Q}$ is the start state of M.

1. Sample a random value $\mathsf{salt} \leftarrow \{0,1\}^{\lambda}$.
2. Interpret $\mathsf{MSK} = (1\mathsf{FE}_1.\mathsf{MSK}, (1\mathsf{FE}_2.\mathsf{PK}, 1\mathsf{FE}_2.\mathsf{MSK}))$.
3. Let $\mathsf{SK}_{\mathsf{ReRand}} = 1\mathsf{FE}_1.\mathsf{KeyGen}(1\mathsf{FE}_1.\mathsf{MSK}, \mathsf{ReRand}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q_{st}},\perp,\perp})$ where Figure 2 defines the circuit $\mathsf{ReRand}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q_{st}},\perp,\perp}$.

Function $\mathsf{ReRand}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q_{st}},\mathcal{C}_1,\mathcal{C}_2}\big((\mathsf{K}_0, i, \ell, w_i, \mathsf{Trap})\big)$

(a) **Initialization and Choosing Real or Trapdoor mode.**
Initialize an input vector $\mathsf{inp} = (w_i, \mathsf{q_{st}})$. If $\mathsf{Trap.mode\text{-}real} = 1$, set $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. If $i \neq 1$, set $\mathsf{inp} = (w_i, \perp)$. Else invoke $\mathsf{Trap\text{-}Mode}_{\mathsf{ReRand}}\big(\mathsf{Trap}, \mathsf{inp}, \mathsf{salt}, \ell, \mathcal{C}_1, \mathcal{C}_2, i\big)$ as described in Figure 3 to obtain $\big(\mathsf{inp} = (u_1, u_2), \mathsf{out} = (c_1, c_2)\big)$.

(b) **Computing Encrypted Symbols using randomness derived from** cPRF. If $\mathsf{out}.c_1 = \perp$, do the following.
  i. Noting that $i > 0$, derive delegated cPRF key K_i from K_0 as $\mathsf{K}_i = \mathsf{F.KeyDel}(\mathsf{K}_0, f_i)$. Compute randomness for encryption as $r_i = \mathsf{F.Eval}(\mathsf{K}_i, (i \| \mathsf{salt}))$.
  ii. Derive delegated cPRF key $\mathsf{K}_{i+1} = \mathsf{F.KeyDel}(\mathsf{K}_i, f_{i+1})$. Set $\mathsf{key\text{-}id} = \mathsf{salt}$.
  iii. Compute the $1\mathsf{FE}_2$ ciphertext component encoding $w_i = \mathsf{inp}.u_1$ for time step i as

$$\mathsf{CT}_{\mathsf{sym},i} = \bar{\mathcal{E}}\big(1\mathsf{FE}_2.\mathsf{PK}_1, (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{i+1}, i, \ell, w_i, \mathsf{Trap}); r_i\big)$$

  iv. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},i}$.

(c) **Computing Encrypted State for First Time Step.** If $\big((\mathsf{out}.c_2 = \perp) \wedge (i = 1)\big)$, do the following.
  i. Compute $1\mathsf{FE}_2$ ciphertext component to encode the starting state $\mathsf{q_{st}} = \mathsf{inp}.u_2$ as

$$\mathsf{CT}_{\mathsf{st},1} = \bar{\mathcal{E}}\big(1\mathsf{FE}_2.\mathsf{PK}_2, (\mathsf{ST}, \mathsf{q_{st}}); r_1\big)$$

  ii. Set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},1}$.

(d) **Output.** If $i = 1$, output $\mathsf{out} = (\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$, else output $\mathsf{out} = (\mathsf{CT}_{\mathsf{sym},i}, \perp)$.

Fig. 2. This circuit re-randomizes the ciphertexts provided during encryption to use randomness derived from a cPRF. The seed for the cPRF is specified in the ciphertext and the input is specified by the key. This ensures that each ciphertext, key pair form a unique "thread" of execution.

4. Let $\mathsf{SK}_{\mathsf{Next}} = 1\mathsf{FE}_2.\mathsf{KeyGen}(1\mathsf{FE}_2.\mathsf{MSK}, \mathsf{Next}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},M,\perp,\perp})$ where Figure 4 defines the circuit $\mathsf{Next}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},M,\perp,\perp}$.
5. Output $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$.

$\mathsf{TMFE.Dec}(\mathsf{SK}_M, \mathsf{CT}_{\mathbf{w}})$: Upon input secret key SK_M and ciphertext $\mathsf{CT}_{\mathbf{w}}$, do the following:

1. Interpret $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ and $\mathsf{CT}_{\mathbf{w}} = \big(\mathsf{CT}_1, \ldots, \mathsf{CT}_{|\mathbf{w}|}\big)$.
2. For $i \in [|\mathbf{w}|]$, do the following:
   (a) If $i = 1$, invoke $1\mathsf{FE}_1.\mathsf{Dec}(\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{CT}_1)$ to obtain $(\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$.
   (b) Else, invoke $1\mathsf{FE}_1.\mathsf{Dec}(\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{CT}_i)$ to obtain $(\mathsf{CT}_{\mathsf{sym},i}, \perp)$.
3. Denote $\big((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \mathsf{CT}_{\mathsf{sym},2}, \ldots, \mathsf{CT}_{\mathsf{sym},|\mathbf{w}|}\big)$ as the new sequence of ciphertexts obtained under the Next scheme.
4. Let $t = 1$. While the Turing machine does not halt, do:
   (a) Invoke $1\mathsf{FE}_2.\mathsf{Dec}\big(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},t}, \mathsf{CT}_{\mathsf{st},t})\big)$ to obtain:
      – ACC or REJ. In this case, output "Accept" or "Reject" respectively, and exit the loop.
      – $\big(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1}\big)$.

Subroutine $\mathsf{Trap\text{-}Mode}_{\mathsf{ReRand}}(\mathsf{Trap}, \mathsf{inp}, \mathsf{salt}, \ell, \mathcal{C}_1, \mathcal{C}_2, i)$

Interpret $\mathsf{inp} = (u_1, u_2) = (w_i, \mathsf{q}_{\mathsf{st}})$ and initialize $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$.

If $\mathsf{Trap.key\text{-}id} = \mathsf{salt}$, do the following.

(a) If $\mathsf{Trap.mode\text{-}trap}_3 = 1$, do the following:
  i. If $((\mathsf{Trap.Sym\ TS} = i) \wedge (i \leq \ell))$, compute the $1\mathsf{FE}_2$ ciphertext $\mathsf{CT}_{\mathsf{sym},i} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_1)$ and set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},i}$.
  ii. If $((\mathsf{Trap.ST\ \ TS} = i) \wedge (i = 1))$, compute the $1\mathsf{FE}_2$ ciphertext $\mathsf{CT}_{\mathsf{st},i} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},1}$.
(b) If $\mathsf{Trap.mode\text{-}trap}_1 = 1$, do the following:
  i. If $((\mathsf{Trap.Sym\ TS}_1 = i) \wedge (i \leq \ell))$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol to be encrypted and output at time step i.
  ii. If $((\mathsf{Trap.ST\ TS}_1 = i) \wedge (i = 1))$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_1$ with the start state to be encrypted and output at time step 1.
(c) If $\mathsf{Trap.mode\text{-}trap}_2 = 1$, do the following:
  i. If $((\mathsf{Trap.Sym\ TS}_2 = i) \wedge (i \leq \ell))$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol to be encrypted and output at time step i.
  ii. If $((\mathsf{Trap.ST\ TS}_2 = i) \wedge (i = 1))$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_2$ with the start state to be encrypted and output at time step 1.

If $\mathsf{Trap.key\text{-}id} \neq \mathsf{salt}$, do the following.

(a) If $\mathsf{salt} > \mathsf{Trap.key\text{-}id}$ set $b = 0$; else set $b = 1$[a].
(b) If $i \neq 1$, update $\mathsf{inp} = (\mathsf{Trap.val}_b, \perp)$; else update $\mathsf{inp} = (\mathsf{Trap.val}_b, \mathsf{q}_{\mathsf{st}})$.

Output. Return $(\mathsf{inp}, \mathsf{out})$.

[a] We assume a lexicographic ordering on the salt values and a generalized comparison operator.

Fig. 3. Subroutine handling the trapdoor modes in ReRand. This is "active" only in the proof.

Note that t' is the next time step that the work tape cell accessed at time step t will be accessed again.

(b) Let $t = t + 1$ and go to start of loop.

3.2 Correctness and Efficiency of single input TMFE

We now argue that the above scheme is correct. The TMFE.Dec algorithm takes as input a secret key $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ and a ciphertext $\mathsf{CT}_{\mathbf{w}} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_{|\mathbf{w}|})$ under the $1\mathsf{FE}_1$ scheme supporting the functionality $\mathsf{ReRand} := \mathsf{ReRand}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\mathcal{C}_2,\mathcal{C}_2}$. Firstly, note that given a secret key $\mathsf{SK}_{\mathsf{ReRand}}$ along with a ciphertext $\mathsf{CT}_{\mathbf{w}}$, we have as follows.

1. Since CT_1 encodes Trap with $\mathsf{Trap.mode\text{-}real} = 1$, hence by the correctness of the $1\mathsf{FE}_1$ scheme, we get $1\mathsf{FE}_1.\mathsf{Dec}(\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{CT}_1) = (\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$ as output.
2. For $i \in [2, |\mathbf{w}|]$, since CT_i encodes Trap with $\mathsf{Trap.mode\text{-}real} = 1$, hence by the correctness of the $1\mathsf{FE}_1$ scheme, we get $1\mathsf{FE}_1.\mathsf{Dec}(\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{CT}_i) = (\mathsf{CT}_{\mathsf{sym},i}, \perp)$ as the correct output.

The new sequence of $1\mathsf{FE}_2$ ciphertexts output by ReRand are now sequenced as $((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \mathsf{CT}_{\mathsf{sym},2}, \ldots, \mathsf{CT}_{\mathsf{sym},|\mathbf{w}|})$. The $1\mathsf{FE}_2$ scheme supports the functionality $\mathsf{Next} := \mathsf{Next}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},M,\mathcal{C}_1,\mathcal{C}_2}$. Throughout the $1\mathsf{FE}_2$ decryption, we maintain the invariant that at any time step t, apart from a secret key $\mathsf{SK}_{\mathsf{Next}}$, the input to the $1\mathsf{FE}_2.\mathsf{Dec}$ algorithm is an entire $1\mathsf{FE}_2$ ciphertext decomposed into two

Function $\mathsf{Next}_{1\mathsf{FE}_2.\mathsf{PK},\mathsf{salt},M,\mathcal{C}_1,\mathcal{C}_2}\big((\mathbf{z}_1,\mathbf{z}_2)\big)$

(a) **Reading Current (Symbol, State) Pair and Looking up Transition Table.**
   i. Interpret $\mathbf{z}_1 = (\mathsf{type}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, s, \mathsf{Trap})$, $\mathbf{z}_2 = (\mathsf{type}, s)$. If $((\mathbf{z}_1.\mathsf{type} \neq \mathsf{SYM}) \vee (\mathbf{z}_2.\mathsf{type} \neq \mathsf{ST}) \vee (\mathbf{z}_1.\mathsf{key\text{-}id} \neq \mathsf{salt}))$, output $\perp$ and abort.
   ii. Interpret $(\mathbf{z}_1.s, \mathbf{z}_2.s) = (\sigma_t, \mathsf{q}_t)$ as the symbol, state pair for the current time step $t = \mathbf{z}_1.t$, input $\mathsf{K}_{t+1} = \mathbf{z}_1.\mathsf{K}_{t+1}$ as the constrained PRF key for future time steps. Denote $\mathsf{key\text{-}id} = \mathbf{z}_1.\mathsf{key\text{-}id}$, $\ell = \mathbf{z}_1.\ell$ and $\mathsf{Trap} = \mathbf{z}_1.\mathsf{Trap}$. Using the transition table of the machine M, look up the next state q_{t+1} as well as the symbol $\sigma_{t'}$ to be written on the work-tape, where t' is the time step the current work tape cell will next be read by M. If q_{t+1} is an accept or reject state, then output ACC or REJ and exit.
   iii. Initialize $\mathsf{inp} = (\sigma_{t'}, \mathsf{q}_{t+1})$.

(b) **Choosing Real or Trapdoor mode.** If $\mathsf{Trap.mode\text{-}real} = 1$, initialize an output vector $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. Else invoke $\mathsf{Trap\text{-}Mode}_{\mathsf{Next}}(\mathsf{Trap}, \mathsf{inp}, \mathsf{salt}, \ell, \mathcal{C}_1, \mathcal{C}_2, t, t')$ as described in Figure 5 to obtain $\big(\mathsf{inp} = (u_1, u_2), \mathsf{out} = (c_1, c_2)\big)$.

(c) **Computing Next Encrypted Symbol.** If $\mathsf{out}.c_1 = \perp$, do the following.
   i. Noting that $t' > t$, derive the randomness at time step t' using the delegated key K_{t+1} as $r_{t'} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t' \| \mathsf{salt}))$. Compute the delegated PRF key $\mathsf{K}_{t'+1} = \mathsf{F.KeyDel}(\mathsf{K}_{t+1}, f_{t'+1})$.
   ii. Compute the $1\mathsf{FE}_2$ ciphertext component encoding the symbol $\sigma_{t'} = \mathsf{inp}.u_1$ for time step t' as
$$\mathsf{CT}_{\mathsf{sym},t'} = \bar{\mathcal{E}}\big(1\mathsf{FE}_2.\mathsf{PK}_1, (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t'+1}, t', \ell, \sigma_{t'}, \mathsf{Trap}); r_{t'}\big)$$
   iii. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.

(d) **Computing Next Encrypted State.** If $\mathsf{out}.c_2 = \perp$, do the following.
   i. Derive the randomness at time step $t+1$ as $r_{t+1} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t+1 \| \mathsf{salt}))$ and compute the $1\mathsf{FE}_2$ ciphertext component encoding the state $\mathsf{q}_{t+1} = \mathsf{inp}.u_2$ for time step $t+1$ as
$$\mathsf{CT}_{\mathsf{st},t+1} = \bar{\mathcal{E}}\big(1\mathsf{FE}_2.\mathsf{PK}_2, (\mathsf{ST}, \mathsf{q}_{t+1}); r_{t+1}\big)$$
   ii. Set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},t+1}$.

(e) **Output :** $\mathsf{out} = \big(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1}\big)$

Fig. 4. Function to mimic TM computation. It reads the current symbol, state pair and outputs an encryption of the new state and symbol to be written under the appropriate randomness generated using a cPRF.

Subroutine $\mathsf{Trap\text{-}Mode}_{\mathsf{Next}}(\mathsf{Trap}, \mathsf{inp}, \mathsf{salt}, \ell, \mathcal{C}_1, \mathcal{C}_2, t, t')$

Interpret the input vector $\mathsf{inp} = (u_1, u_2) = (\sigma_{t'}, \mathsf{q}_{t+1})$ and initialize the output vector $\mathsf{out} = (c_2, c_2)$, where $c_1 = c_2 = \perp$.

(a) If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.mode\text{-}trap}_3 = 1)\big)$, do the following.
   i. If $\big((\mathsf{Trap.Sym\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t') \wedge (t > \ell)\big)$, compute the $1\mathsf{FE}_2$ symbol ciphertext $\mathsf{CT}_{\mathsf{sym},t'} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_1)$ and set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.
   ii. If $\big((\mathsf{Trap.ST\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t+1) \wedge (t > 1)\big)$, compute the $1\mathsf{FE}_2$ state ciphertext $\mathsf{CT}_{\mathsf{st},t+1} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},t+1}$.

(b) If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.mode\text{-}trap}_1 = 1)\big)$, do the following:
   i. If $\big((\mathsf{Trap.Sym\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_1$ to be encrypted and given as output for time step t'.
   ii. If $\big((\mathsf{Trap.ST\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_1$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_1$ to be encrypted and given as output for time step $t+1$.

(c) If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.mode\text{-}trap}_2 = 1)\big)$, do the following:
   i. If $\big((\mathsf{Trap.Sym\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_2$ to be encrypted and given as output for time step t'.
   ii. If $\big((\mathsf{Trap.ST\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_2$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_2$ to be encrypted and given as output for time step $t+1$.

(d) Exit the subroutine returning $(\mathsf{inp}, \mathsf{out})$.

Fig. 5. Subroutine handling the trapdoor modes in Next. This is "active" only in the proof.

components corresponding to a symbol and a state ciphertext both of which are computed with the same randomness, which is computed as $\mathsf{F.Eval}(\mathsf{K}_0, (\mathsf{t} \| \mathsf{salt}))$[8].

[8] We do not explicitly construct ciphertext components corresponding to blank tape cells in the Next functionality for ease of exposition; we assume w.l.o.g that any non-input cell that is accessed by the OTM has been written to by the Next functionality in a previous step, thus generating the requisite symbol ciphertext.

We show that given a secret key $\mathsf{SK}_{\mathsf{Next}}$ and the sequence of ciphertexts $\big((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \mathsf{CT}_{\mathsf{sym},2}, \ldots, \mathsf{CT}_{\mathsf{sym},|\mathbf{w}|}\big)$ generated from the outputs of the $1\mathsf{FE}_1.\mathsf{Dec}$ algorithm, $1\mathsf{FE}_2.\mathsf{Dec}$ correctly computes the decomposed ciphertext components of a symbol and a state that occur along the computation path and finally outputs the value of machine M on the sequenced input. Define $\tau = \mathsf{runtime}(M, \mathbf{w})$. Formally, by the correctness of $1\mathsf{FE}_2$ scheme, at any time step $t \in [\tau - 2]$, $1\mathsf{FE}_2.\mathsf{Dec}(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},t}, \mathsf{CT}_{\mathsf{st},t}))$ correctly outputs either $(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1})$ with $t < t' \leq \tau - 1$. Further, for any time step $t \in [\tau - 2]$, we have:

1. Let $t \in [\tau - 2] \setminus [\ell]$. If the current work tape cell was accessed[9], at some time step $\tilde{t} < t$, then $\mathsf{CT}_{\mathsf{sym},t}$ encoding $(\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, \sigma_t, \mathsf{Trap})$ was constructed at time step $\tilde{t}$. Note that σ_t may be the blank symbol β. When $t \in [\ell]$, $\mathsf{CT}_{\mathsf{sym},t}$ is constructed at time step t via the ReRand circuit.
2. The ciphertext component $\mathsf{CT}_{\mathsf{st},t}$ encoding $(\mathsf{ST}, \mathsf{q}_t)$ at time step t was constructed at time step $t - 1$ for $t > 1$ and at time step 1, when $t = 1$.
3. The randomness $r_t = \mathsf{F.Eval}(\mathsf{K}_{\tilde{t}+1}, (\mathsf{t}\|\mathsf{salt})) = \mathsf{F.Eval}(\mathsf{K}_{\mathsf{t}}, (\mathsf{t}\|\mathsf{salt}))$ binds the components $\mathsf{CT}_{\mathsf{sym},t}$ and $\mathsf{CT}_{\mathsf{st},t}$.

Thus, at any given time step $t \in [\tau - 2]$, we have a complete ciphertext of $1\mathsf{FE}_2$ which may be fed again with $\mathsf{SK}_{\mathsf{Next}}$ to $1\mathsf{FE}_2.\mathsf{Dec}$ in order to proceed with the computation. Thus, the execution of $1\mathsf{FE}_2.\mathsf{Dec}$ at the $(\tau - 2)^{\text{th}}$ time step provides the complete pair $(\mathsf{CT}_{\mathsf{sym},\tau-1}, \mathsf{CT}_{\mathsf{st},\tau-1})$. By the correctness of $1\mathsf{FE}_2$ scheme again, at time step $t = \tau - 1$, invoking $1\mathsf{FE}_2.\mathsf{Dec}(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},\tau-1}, \mathsf{CT}_{\mathsf{st},\tau-1}))$ outputs either "Accept" or "Reject" by simulating the execution of M for the final time step τ inside the function Next, thus correctly outputting $M(\mathbf{w})$.

Efficiency. The TMFE construction described above inherits its efficiency from the underlying CktFE constructions. Note that the ciphertext is compact and is of size $\mathrm{poly}(\lambda, |\mathbf{w}|)$. Also, the running time of the decryption procedure is input specific since it mimics the computation of M on $\mathbf{w}$ using secret key encoding M and ciphertext encoding all the intermediate states of the computation. Additionally, the public parameters are short $\mathrm{poly}(\lambda)$, since these are just the public parameters of a compact CktFE scheme. The function keys are also short, since they are CktFE function keys for circuits ReRand and Next which are of size $\mathrm{poly}(\lambda)$ and $\mathrm{poly}(|M|, \lambda)$ respectively.

3.3 Proof of Security for Single Input TMFE

Next, we prove that the above TMFE scheme satisfies distributional indistinguishability (DI) for single (or constant) length outputs, as long as the underlying CktFE scheme satisfies distributional indistinguishability for any output length. In the full version [1], we provide an instantiation of a CktFE scheme satisfying distributional indistinguishability.

Theorem 1. *Assume that the functional encryption schemes for circuits* $1\mathsf{FE}_1$ *and* $1\mathsf{FE}_2$ *are* DI *secure and that* F *is a secure* cPRF *for the function family defined above. Then, the construction of functional encryption for Turing machines* TMFE *is selective* DI *secure for single bit outputs.*

The Trapdoor Data Structure. To implement the approach discussed in Section 1, we will make use of a data-structure Trap that lets us store all the requisite trapdoor information needed for the security proof within the ciphertext. In our construction, decryption of a particular input by a particular function key results in a chain of ciphertexts, each of which contain the trapdoor data structure. In the real world, this information is not used but as we progress through the proof, different fields become relevant. The data structure is outlined in Figure 6.

[9] We assume that every time a cell is accessed, it is written to, by writing the same symbol again if no change is made.

mode-real	key-id	val_0	val_1	SKE.K	$\perp$
$\mathsf{mode\text{-}trap}_1$	Target TS_1	Sym TS_1	Sym val_1	ST TS_1	ST val_1
$\mathsf{mode\text{-}trap}_2$	Target TS_2	Sym TS_2	Sym val_2	ST TS_2	ST val_2
$\mathsf{mode\text{-}trap}_3$	Target TS	Sym TS	$\perp$	ST TS	$\perp$

Fig. 6. Data Structure Trap used for Proof

Row 1. Above, key-id refers to the particular function key being considered and we switch the execution chain from $b = 0$ to $b = 1$ key by key. All the ciphertexts in a given execution chain share the key-id value. We assume a lexicographic order on the key-id fields, this can be easily ensured by having a counter as part of the key-id field. We do not make this explicit below for notational brevity. If key-id* is the key identity programmed in a particular execution chain, then all keys with values smaller than key-id* will decrypt the chain using the input bit $b = 1$, and all keys with values larger than key-id* will use $b = 0$. Hence, the $1\mathsf{FE}_1$ ciphertexts provided by the encryptor must encode messages corresponding to both values of b, the fields val_0 and val_1 are designed for this purpose[10]. Note that $1\mathsf{FE}_2$ ciphertexts computed by decryption need not track messages corresponding to both values of b, since the "chain is extended" via decryption corresponding to exactly one of $b = 0$ or $b = 1$ depending on the relation between the key identities in the ciphertext and the function key. The field SKE.K refers to the key of a symmetric key encryption scheme, which is used to decrypt some encrypted value embedded in the function key. This is a standard trick when the key must hide something in the public key setting. The flag mode-real means the scheme operates in the real world mode and the trapdoor information is not used.

Rows 2 and 3. The fields Target TS_1 and Target TS_2 refer to the time steps corresponding to the "broken link" in the decryption chain, namely the two time steps for which the ciphertext and function key are being programmed so as to switch from $b = 0$ to $b = 1$. The fields Sym TS_1 and ST TS_1 are the time steps when the symbol and state ciphertexts for time step Target TS_1 are generated; for instance ST TS_1 = Target $\mathsf{TS}_1 - 1$ since the state ciphertext for a given time step is always generated in the previous time step, while the symbol ciphertext for a given time step may be generated much earlier. Sym TS_2 and ST TS_2 are defined analogously. The fields Sym val_1 and ST val_1 contain the symbol and state values which will be encrypted in the hybrid at the time steps Sym TS_1 and ST TS_1 when $\mathsf{mode\text{-}trap}_1$ is set; Sym val_2 and ST val_2 are defined analogously.

Row 4. When $\mathsf{mode\text{-}trap}_3$ is set, the symbol and state values are set to $\perp$, and the values hard coded in the function key are used for the target time step. In more detail, the function key contains SKE encryptions of symbol and state ciphertexts corresponding to time step Target TS hard-coded within itself. If key-id* = key-id, where key-id* is the key identity programmed in a particular execution chain and key-id is the key identity of the function key in question, and $\mathsf{mode\text{-}trap}_3 = 1$, then at time steps SYM TS and ST TS the SKE secret key in row 1 of the Trap data structure is used to decrypt the SKE encryptions and output the encrypted values.

The Hybrids. We now proceed to describe our hybrids. For simplicity we first describe the hybrids for a single function request, for some Turing machine M. We denote by T the time taken by M to run on the challenge messages. Since the proof is very involved, we describe it first for the weak selective game, where the adversary specifies the challenge vectors and machine at the same time. In the full version [1] we discuss how to remove this restriction.

$\mathcal{H}(0)$: This is the real world, when mode-real $= 1$ and $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = \mathsf{mode\text{-}trap}_3 = \perp$.

$\mathcal{H}(1,1)$: In this world, all ciphertexts (constructed by the encryptor as well as function keys) have mode-real $= \perp$, $\mathsf{mode\text{-}trap}_1 = 1,\ \mathsf{mode\text{-}trap}_2 = 1,\ \mathsf{mode\text{-}trap}_3 = \perp$. We program the last link in

[10] For the knowledgeable reader, this is similar to what was done by [5].

the decryption chain for switching bit b by setting:

$$\mathsf{Target\ TS}_1 = T-1, \mathsf{Target\ TS}_2 = T-2$$

The fields $\mathsf{Sym\ TS}_1$ and $\mathsf{ST\ TS}_1$ contain the time steps when the symbol and state ciphertext pieces are generated for time step $T-1$, and the fields $\mathsf{Sym\ val}_1$ and $\mathsf{ST\ val}_1$ contain the symbol and state values which must be encrypted by the function key in the above time steps when $\mathsf{mode\text{-}trap}_1$ is set. Note that these fields exactly mimic the behaviour in the real world, namely the time steps and values are set to be exactly what the real world decryption would output. The fields corresponding to TS_2 are defined analogously.

Indistinguishability follows from security of $\mathsf{1FE}_1$, since the decryption values in both hybrids are exactly the same.

$\mathcal{H}(1,2)$: Hardwire the key with an SKE encryption of symbol and state ciphertexts output at step $T-1$ for $b=0$. Use the same ciphertexts as would be generated in the previous hybrid.

Indistinguishability follows from security of SKE, since the only difference is the value of the message encrypted using SKE which is embedded in the key.

$\mathcal{H}(1,3)$: Set $\mathsf{mode\text{-}trap}_1 = \perp$, $\mathsf{mode\text{-}trap}_2 = 1$, $\mathsf{mode\text{-}trap}_3 = 1$ and $\mathsf{Target\ TS} = T-1$. In this hybrid the hardwired value in the key is used to be output as step $T-1$ ciphertext.

Indistinguishability follows from security of $\mathsf{1FE}_1$, since the decryption values in both hybrids are exactly the same.

$\mathcal{H}(1,4)$: Change normal root key K_0 to punctured root key K_0^{T-1} which punctures all delegated keys at point $(T-1||\mathsf{key\text{-}id})$.

Indistinguishability follows from security of $\mathsf{1FE}_1$. Note that we evaluate the cPRF at point $(T-1||\mathsf{key\text{-}id})$ only to construct the $\mathsf{1FE}_2$ ciphertext output at time step $T-1$ identified with $\mathsf{key\text{-}id}$. This ciphertext is currently hardwired in the function key, and is computed exactly the same way in both hybrids. Thus, the cPRF key is only required to compute randomness of points $\neq (T-1||\mathsf{key\text{-}id})$, for which the punctured key suffices, and which moreover evaluates to the same value as the normal key on all such points. Hence, we have that the decryption values in both hybrids are exactly the same. Note that the punctured key is not used to evaluate on the punctured points.

$\mathcal{H}(1,5)$: Switch the randomness in the $\mathsf{1FE}_2$ ciphertexts for time step $T-1$ which are hardwired in the key to true randomness.

Indistinguishability follows from security of punctured cPRF for the aforementioned function family, since the remainder of the distribution only uses the punctured key.

$\mathcal{H}(1,6)$: Switch the value encoded in the $\mathsf{1FE}_2$ ciphertexts for time step $T-1$ which are hardwired in the key to correspond to $b=1$.

Indistinguishability follows from security of $\mathsf{1FE}_2$. Formally, we do a reduction which plays the security game against the $\mathsf{1FE}_2$ challenger and simulates the TMFE adversary. The reduction simulates $\mathsf{1FE}_1$ itself and receives the $\mathsf{1FE}_2$ public and function keys from the challenger. The only difference between the two hybrids is the $\mathsf{1FE}_2$ ciphertext for time step $T-1$ which is embedded in the function key as received from the $\mathsf{1FE}_2$ challenger.

$\mathcal{H}(1,7)$: Switch randomness back to PRF randomness in the ciphertext hardwired in key, using the punctured key for all but the hardwired ciphertext.

Indistinguishability follows from security of cPRF as discussed above.

$\mathcal{H}(1,8)$: Switch the punctured root key to the normal root key.

Indistinguishability follows from security of $\mathsf{1FE}_1$ as discussed above.

$\mathcal{H}(2,1)$: Switch ciphertext in slot 1 for target $T-1$ to be for $b=1$. Slot 2 remains $b=0$. Set $\mathsf{mode\text{-}trap}_3 = \perp$ and $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = 1$.

Indistinguishability follows from security of $\mathsf{1FE}_1$, since the decryption values in both hybrids are exactly the same.

$\mathcal{H}(2,2)$: Hardwire key with SKE encryption of $\mathsf{1FE}_2$ ciphertext for time step $T-2$ and bit $b=0$ (same as hybrid $(1,2)$ but for $T-2$).
Indistinguishability follows from security of SKE as above.

$\mathcal{H}(2,3)$: Set $\mathsf{mode\text{-}trap}_1 = 1$ with target $T-1$, $\mathsf{mode\text{-}trap}_2 = \perp$, and $\mathsf{mode\text{-}trap}_3 = 1$ with target $T-2$.
Indistinguishability follows from security of $\mathsf{1FE}_1$, since the decryption values in both hybrids are exactly the same.

$\mathcal{H}(2,4)$: Switch normal root key to punctured key at point $(T-2\|\mathsf{key\text{-}id})$.
Indistinguishability follows from security of $\mathsf{1FE}_1$ as discussed above.

$\mathcal{H}(2,5)$: Switch randomness to true in the ciphertext hardwired in key.
Indistinguishability follows from security of cPRF as discussed above.

$\mathcal{H}(2,6)$: Switch hardwired $\mathsf{1FE}_2$ ciphertext for step $T-2$ to correspond to bit $b=1$.
Indistinguishability follows from security of $\mathsf{1FE}_2$.

$\mathcal{H}(2,7)$: Switch randomness back to use the PRF in the ciphertext hardwired in key.
Indistinguishability follows from security of cPRF as discussed above.

$\mathcal{H}(2,8)$: Switch punctured root key to normal root key.
Indistinguishability follows from security of $\mathsf{1FE}_1$ as discussed above.

$\mathcal{H}(3,1)$: Intuitively, we slide the trapdoor left by one step, i.e. change target time-steps to $T-2$ and $T-3$ in the ciphertext. Now slot 1 for $T-2$ corresponds to $b=1$ and slot 2 for $T-3$ to $b=0$. Set $\mathsf{mode\text{-}real} = \mathsf{mode\text{-}trap}_3 = \perp$ and $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = 1$.
Indistinguishability follows from security of $\mathsf{1FE}_1$, since the decryption values in both hybrids are exactly the same. Note that now slot $T-1$ is redundant, since $T-2$ ciphertext is already switched to $b=1$.
Hybrid $\mathcal{H}(3,i)$ will be analogous to $\mathcal{H}(2,i)$ for $i \in [8]$.
As we proceed left in the execution chain one step at a time, we reach step ℓ where $\ell = |\mathbf{w}|$, i.e. time steps for which $\mathsf{1FE}_1$ ciphertexts are provided by the encryptor. At this point we will hardwire the ReRand key with symbol ciphertexts for ℓ time steps, one at a time, and the Next key for the state ciphertexts[11]. Moreover, we must now add an additional hybrid in which the challenge $\mathsf{1FE}_1$ ciphertext at position ℓ contains the message bit corresponding to $b=1$; intuitively, we must switch the bit before we slide the trapdoor since the ciphertext for this position is not generated by decrypting the previous ciphertext. In more detail, in $\mathcal{H}(T-\ell,8)$, analogously to hybrid $(1,8)$, the $T-(T-\ell) = \ell^{th}$ bit hard-wired in the trapdoor is changed to 1. We now add one more hybrid, namely:

$\mathcal{H}(T-\ell,9)$: In this hybrid, we modify the $\mathsf{1FE}_1$ challenge ciphertext in position ℓ as follows: the encoded message is changed corresponding to $b=1$ and flag $\mathsf{mode\text{-}real} = 1$. The other flags $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = \mathsf{mode\text{-}trap}_3 = \perp$.
Note that all ciphertexts previous to time step ℓ remain unchanged, and output their corresponding symbol ciphertexts correctly. The Next circuit outputs the state ciphertext for time step ℓ corresponding to bit $b=1$. The only difference between this hybrid and the previous one is that here we use the real mode to output the symbol ciphertext for $b=1$ whereas previously we used the trapdoor mode to output the same symbol ciphertext. Hence, decryption values in both hybrids are exactly the same, and indistinguishability follows from security of $\mathsf{1FE}_1$.
Finally in $\mathcal{H}(T-1,9)$, the entire chain has been replaced to use $b=1$ and all the challenge $\mathsf{1FE}_1$ ciphertexts have encoded messages corresponding to $b=1$ with $\mathsf{mode\text{-}real} = 1$.

$\mathcal{H}(T)$: In this hybrid, all the other fields in the trapdoor data structure, excepting $\mathsf{mode\text{-}real}$ are disabled and set to $\perp$. This is the real world with $b=1$.
Since all the encoded messages use $b=1$, decryption values are all exactly the same as in $\mathcal{H}(T-1,9)$, hence indistinguishability follows from security of $\mathsf{1FE}_1$.

The formal reductions are provided in the full version [1].

[11] There is an exception at time step 1 when both the symbol ciphertext and the start state ciphertexts are hardwired in the ReRand key

Multiple Keys. We handle multiple keys by repeating the above set of hybrids key by key. Each key carries within it an identifier key-id, and if this is less than the key identifier encoded in the ciphertext, the bit $b = 1$ is used, if it is greater then the bit $b = 0$ is used and if it is equal, then the above sequence of hybrids is performed to switch from $b = 0$ to $b = 1$. To support this, the $1\mathsf{FE}_1$ ciphertexts provided by the encryptor must encode messages corresponding to both values of b, the fields val_0 and val_1 in the trapdoor data structure of Figure 6 are provided for this purpose. Security follows by a standard hybrid argument as in [5], we defer the formal description to the full version of the paper [1].

3.4 Constructing the cPRF.

In the full version [1], we provide a construction for a cPRF F which supports puncturing and delegation as required; the T cPRFs F_i for $i \in [T]$ may each be constructed similarly. To begin, note that we require the root key of F to be punctured at a point i^* (say). The cPRF construction for punctured PRF [17, 36, 18](which is in turn inherited from the standard PRG based GGM [32]) immediately satisfies this constraint, so we are left with the question of delegation.

Recall that we are required to delegate T times, where T is the (polynomial) runtime of the Turing machine on the encrypted input (please see preliminaries in [1]), and the j^{th} delegated key must support evaluation of points $\{(k\|z) : z \in \{0,1\}^\lambda\}$ for $k \geq j$, *except* when $(k\|z) = i^*$. This may be viewed as the j^{th} key being punctured on points $[1, j-1] \cup i^*$. We show that the GGM based construction for puncturing a single point can be extended to puncturing an interval (plus an extra point). Intuitively, puncturing an interval corresponds to puncturing at most λ internal nodes in the GGM tree. In more detail, we show that regardless of the value of j, it suffices to puncture at most λ points in the GGM tree to achieve puncturing of the entire interval $[1, j-1]$. Please see the full version [1] for details.

4 Construction: Multi-Input FE for Turing Machines

In this section we construct a multi-input functional encryption scheme for Turing machines. Our construction supports a fixed number of encryptors (say k), who may each encrypt a string $\mathbf{w}_i$ of *unbounded* length. Function keys may be provided for Turing machines, so that given k ciphertexts for $\mathbf{w}_i$ and a function key for TM M , decryption reveals $M(\mathbf{w}_1\|\ldots\|\mathbf{w}_k)$ and nothing else. We use the following ingredients for our construction:

1. A compact, k-input functional encryption scheme for circuits, kFE and a compact, public-key functional encryption scheme 1FE. As before, we will assume that the scheme 1FE is decomposable as defined in the preliminaries.
2. A symmetric encryption scheme SKE = (SKE.KeyGen, SKE.Enc, SKE.Dec).
3. A delegatable constrained pseudorandom function (cPRF), denoted by F which supports T delegations for the function family $f_t : \{0,1\}^{(k+2)\cdot\lambda} \to \{0,1\}$ defined as follows. Let x, t denote integers whose binary representations are $\mathbf{x}, \mathbf{t}$ of λ bits. Then,

$$f_t(\mathbf{x}\|\mathbf{z}) = 1, \text{if } \ x \geq t \text{ and } 0 \text{ otherwise}$$

The functionalities supported by kFE and 1FE are called Agg and Next respectively, described next. Agg aggregates the inputs $\mathbf{w}_1, \ldots, \mathbf{w}_k$ of all k parties into one long "global" string $(\mathbf{w}_1\|\ldots\|\mathbf{w}_k)$, encrypted under the scheme 1FE. Since the length of this aggregate string is unbounded, a single invocation of Agg produces an encryption of a single symbol in the string, and the function is invoked repeatedly to produce ciphertexts for the entire string. Each ciphertext output by the Agg scheme contains a symbol w_i as well as the position of the symbol within the global string. The encryption of the symbols (and the initial state) also contains a *global salt* which Agg computes from the random salts provided in the ciphertexts under the kFE scheme by the individual encryptors. The global salt identifies the particular input combination that is aggregated, and serves as input to the PRF in the Next functionality.

Our k-input CktFE scheme may be either private or public key, and will result in the corresponding notion for k-input TMFE. Since the multi input setting for FE is considered more interesting in the symmetric key setting (see [19] for a discussion), we present our construction in the symmetric key setting – the public key adaptation is straightforward.

We note that ciphertexts output by Agg, which are encryptions of the symbols in the aggregate string under the 1FE scheme, are exactly the same as the output of the ReRand function in the single input scheme of Section 3. Therefore, as before, we may have the functionality Next of the 1FE scheme mimic the computation of the Turing machine on the global string $(\mathbf{w}_1 \| \ldots \| \mathbf{w}_k)$. As in the previous construction, 1FE.Dec accepts as its inputs a ciphertext decomposed into two components encoding the current symbol on the worktape and the current state in the computation, both of which have been encrypted using the same randomness, and outputs a ciphertext component corresponding to the symbol written on the tape, as well as the next state. The global salt in the ciphertext, along with a random nonce chosen by KeyGen are used as input to a cPRF as before, to compute the randomness used to generate ciphertexts. This ensures that the execution of a given machine on a given input combination is maintained separate from any other execution, and thwarts "mix and match" attacks, where, for instance, an attacker may try to combine a state generated at some time step t in one execution with a symbol generated at time step t from a different execution.

If we instantiate the underlying multi-input CktFE by the construction of [40], we may let the arity k be poly-logarithmic in the security parameter. If we instantiate multi-input CktFE by the construction of [33], we may support fixed polynomial arity at the cost of worsening the assumption. Note that [33] rely on iO while [40] rely on compact FE. Note that [10] support unbounded polynomial arity, but from public coin DiO as discussed in Section 1.

4.1 Construction of multi-input TMFE

In the following, we denote a k-input, private-key CktFE scheme by k-CktFE and a decomposable, public key CktFE scheme by 1FE. Since our scheme supports an a-priori fixed number of parties, say k, we assume that every user is pre-assigned an index ind $\in [k]$.

$\mathsf{kTMFE.Setup}(1^\lambda, 1^k)$: Upon input the security parameter 1^λ and the bound 1^k, do the following:

1. **Choosing the functionality for** 1FE**.** Let 1FE be a decomposable, public-key CktFE for the following circuit family.

$$\mathsf{Next} : \Big((\{\mathsf{SYM}\} \times \{0,1\}^{(k+4)\lambda} \times \Sigma \times \mathsf{Trap}) \times (\{\mathsf{ST}\} \times \mathcal{Q} \times \{0,1\}^{k\cdot\lambda}) \Big) \rightarrow \Big(\mathcal{C}^{\mathsf{1FE}}\Big)^2 \cup \{\mathsf{ACC}, \mathsf{REJ}, \perp\}$$

   The tokens SYM and ST are flags denoting a symbol and a state respectively of a Turing machine M which has Σ and $\mathcal{Q}$ as the alphabet and state space respectively. The set $\{0,1\}^{(k+4)\lambda}$ encodes in order, a random value key-id associated with a TM M, a constrained PRF key, the current time step in the computation, the length of the input string, each of λ bits and a string of length $k\cdot\lambda$ bits encoding a random value gsalt. Here, Trap is a data structure of fixed polynomial length which will be used in the proof. Since we do not need it in the construction, we do not discuss it here, please see the full version [1] for its definition. The set $\{0,1\}^{k\cdot\lambda}$ encodes again a random value gsalt associated with the message component for state. $\mathcal{C}^{\mathsf{1FE}}$ is the ciphertext space of 1FE. ACC and REJ denote tokens when M reaches an accepting state and a rejecting state respectively.

2. **Choosing the functionality for** kFE**.** Let kFE be a k-CktFE for the following circuit family.

$$\mathsf{Agg} : (\{\mathsf{SYM}, \mathsf{SP}\} \times \{0,1\}^{4\lambda} \times [k] \times \Sigma \times \mathsf{Trap})^k \rightarrow \mathcal{C}^{\mathsf{1FE}} \times \Big(\mathcal{C}^{\mathsf{1FE}} \cup \{\perp\}\Big)$$

The special token SP denotes an encryption of the length of an input string corresponding to any user. The set $\{0,1\}^{4\lambda}$ encodes in order, a constrained PRF key, the time step of the current symbol, the input length and a random salt each of λ bits. Σ, Trap and $\mathcal{C}^{\mathsf{1FE}}$ are as described above.

3. **Choosing keys for kFE and $\mathsf{1FE}$.**

$$\text{Let } \mathsf{kFE.MSK} \leftarrow \mathsf{kFE.Setup}(1^\lambda, 1^k), (\mathsf{1FE.PK}, \mathsf{1FE.MSK}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, 1^k)$$

4. Output $\mathsf{MSK} = (\mathsf{kFE.MSK}, (\mathsf{1FE.PK}, \mathsf{1FE.MSK}))$.

$\mathsf{kTMFE.Enc}(\mathsf{MSK}, \mathbf{w}_{\mathsf{ind}}, \mathsf{ind})$: Upon input the master key MSK, and message $\mathbf{w}_{\mathsf{ind}}$ of arbitrary length ℓ_{ind} and an index $\mathsf{ind} \in [k]$, do the following:

1. Interpret the input $\mathsf{MSK} = (\mathsf{kFE.MSK}, (\mathsf{1FE.PK}, \mathsf{1FE.MSK}))$.
2. Let $\mathbf{w}_{\mathsf{ind}} = w_1 w_2 \ldots w_{\ell_{\mathsf{ind}}}$. Sample $\mathsf{salt}_{\mathsf{ind}} \leftarrow \{0,1\}^\lambda$.
3. Construct the data structure Trap and set all its fields to $\perp$ except a flag $\mathsf{Trap.mode\text{-}real} = 1$ which indicates that we are in the real world. The data structure Trap is only relevant in the proof. Please see [1] for the definition of Trap.

- **Encoding Input String and Its Length**

4. If $\mathsf{ind} = 1$, do the following:
   (a) Sample a root key for the constrained PRF F as $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
   (b) Construct the input message $\mathsf{len}_1 = (\mathsf{SP}, \mathsf{K}_0, \perp, \ell_1, \mathsf{salt}_1, 1, \perp, \mathsf{Trap})$.
   (c) Encrypt ℓ_1 as a special ciphertext $\mathsf{CT}_{1,\mathsf{SP}} = \mathsf{kFE.Enc}(\mathsf{kFE.MSK}, \mathsf{len})$.
   (d) For $i \in [\ell_1]$ do the following:
      i. Construct the input message $\mathbf{y}_{1,i} = (\mathsf{SYM}, \mathsf{K}_0, i, \ell_1, \mathsf{salt}_1, 1, w_i, \mathsf{Trap})$.
      ii. Compute the ciphertext $\mathsf{CT}_{1,\mathsf{SYM},i} = \mathsf{kFE.Enc}(\mathsf{kFE.MSK}, \mathbf{y}_i)$.
5. If $\mathsf{ind} \in [2,k]$, do the following:
   (a) Construct the input message $\mathsf{len}_{\mathsf{ind}} = (\mathsf{SP}, \perp, \perp, \ell_{\mathsf{ind}}, \mathsf{salt}_{\mathsf{ind}}, \mathsf{ind}, \perp, \mathsf{Trap})$.
   (b) Encrypt ℓ_{ind} as a special ciphertext $\mathsf{CT}_{\mathsf{ind},\mathsf{SP}} = \mathsf{kFE.Enc}(\mathsf{kFE.MSK}, \mathsf{len})$.
   (c) For $i \in [\ell_{\mathsf{ind}}]$ do the following:
      i. Construct the input message $\mathbf{y}_{\mathsf{ind},i} = (\mathsf{SYM}, \perp, i, \ell_{\mathsf{ind}}, \mathsf{salt}_{\mathsf{ind}}, \mathsf{ind}, w_i, \mathsf{Trap})$.
      ii. Compute the ciphertext $\mathsf{CT}_{\mathsf{ind},\mathsf{SYM},i} = \mathsf{kFE.Enc}(\mathsf{kFE.MSK}, \mathbf{y}_i)$.
6. Output $\mathsf{CT}_{\mathbf{w}_{\mathsf{ind}}} = \left(\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}, \{\mathsf{CT}_{\mathsf{ind},\mathsf{SYM},i}\}_{i \in [\ell_{\mathsf{ind}}]}\right)$.

$\mathsf{kTMFE.KeyGen}(\mathsf{MSK}, M)$: Upon input the master secret key MSK and the description of a Turing machine M, do the following. We will assume, w.l.o.g. that the TM is oblivious (see [1] for a justification) and $\mathsf{q}_{\mathsf{st}} \in \mathcal{Q}$ is the start state of M.

1. Sample a random value $\mathsf{rand} \leftarrow \{0,1\}^\lambda$.
2. Interpret $\mathsf{MSK} = (\mathsf{kFE.MSK}, (\mathsf{1FE.PK}, \mathsf{1FE.MSK}))$.
3. Let $\mathsf{SK}_{\mathsf{Agg}} = \mathsf{kFE.KeyGen}(\mathsf{kFE.MSK}, \mathsf{Agg}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{q}_{\mathsf{st}},\perp,\perp})$, where Figure 7 defines the circuit $\mathsf{Agg}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{q}_{\mathsf{st}},\perp,\perp}$.
4. Let $\mathsf{SK}_{\mathsf{Next}} = \mathsf{1FE.KeyGen}(\mathsf{1FE.MSK}, \mathsf{Next}_{\mathsf{1FE.PK},\mathsf{rand},M,\perp,\perp})$, where Figure 9 defines the circuit $\mathsf{Next}_{\mathsf{1FE.PK},\mathsf{rand},M,\perp,\perp}$.
5. Output the secret key as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{Agg}}, \mathsf{SK}_{\mathsf{Next}})$.

$\mathsf{kTMFE.Dec}(\mathsf{SK}_M, \{\mathsf{CT}_{\mathbf{w}_i}\}_{i \in [k]})$: Upon input secret key SK_M and k ciphertexts $\mathsf{CT}_{\mathbf{w}_1}, \ldots, \mathsf{CT}_{\mathbf{w}_k}$, do the following:

1. Interpret the secret key as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{Agg}}, \mathsf{SK}_{\mathsf{Next}})$.
2. Parse $\mathsf{CT}_{\mathbf{w}_{\mathsf{ind}}} = (\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}, (\mathsf{CT}_{\mathsf{ind},\mathsf{SYM},1}, \ldots, \mathsf{CT}_{\mathsf{ind},\mathsf{SYM},\ell_{\mathsf{ind}}}))$ for all $\mathsf{ind} \in [k]$.

Function $\mathsf{Agg}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{q_{st}},\mathcal{C}_1,\mathcal{C}_2}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$

(a) Interpret $\mathbf{x}_i = (\mathsf{type}, \mathsf{K}, t, \ell, \mathsf{salt}, \mathsf{ind}, s, \mathsf{Trap})$, for $i \in [k]$ and set a flag $\mathsf{proceed}_1 = \mathsf{proceed}_2 = 0$.

(b) For all $i, j \in [k]$, if $\mathbf{x}_i.\mathsf{ind} \neq \mathbf{x}_j.\mathsf{ind}$ for $i \neq j$, set $\mathsf{proceed} = 1$. If there exists *exactly* one $i \in [k]$ for which $\mathbf{x}_i.\mathsf{type} = \mathsf{SYM}$ and $\mathbf{x}_j.\mathsf{type} = \mathsf{SP}, \forall j \in [k] \setminus \{i\}$ and $\mathsf{proceed}_1 = 1$, set $\mathsf{proceed}_2 = 1$. If $\mathsf{proceed}_2 = 0$, output $\perp$ and abort.

(c) **Initialization and Choosing Real or Trapdoor mode.**
Let $i \in [k]$ be such that $\mathbf{x}_i.\mathsf{type} = \mathsf{SYM}$. Initialize an input vector $\mathsf{inp} = (\sigma, \mathsf{q_{st}})$, where $\sigma = \mathbf{x}_i.s$. Let $\mathsf{gsalt} = (\mathbf{x}_1.\mathsf{salt} \| \mathbf{x}_2.\mathsf{salt} \| \ldots \| \mathbf{x}_k.\mathsf{salt})$ and $\ell = \sum_{i=1}^{k} \mathbf{x}_i.\ell$ denote the global salt and the aggregate input length respectively. Denote $\mathsf{pos} = \mathbf{x}_i.t$ and do the following:

   i. **Computing Global Symbol Position :** If $1 < \mathbf{x}_i.\mathsf{ind} \leq k$, compute the new position of the symbol as $\mathsf{pos} = \mathsf{pos} + \sum_{r \in \mathcal{S}} \mathbf{x}_r.\ell$, where the set $\mathcal{S} = \{r \mid \mathbf{x}_r.\mathsf{ind} < \mathbf{x}_i.\mathsf{ind}\} \subset [k]$.

   ii. If $\mathsf{Trap.mode\text{-}real} = 1$, set $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. If $\mathsf{pos} \neq 1$, set $\mathsf{inp} = (\sigma, \perp)$.

   iii. Else obtain $(\mathsf{inp} = (u_1, u_2), \mathsf{out} = (c_1, c_2)) = \mathsf{Trap\text{-}Mode}_{\mathsf{Agg}}(\mathsf{Trap}, \mathsf{inp}, \mathsf{rand}, \mathsf{gsalt}, \ell, \mathcal{C}_1, \mathcal{C}_2, \mathsf{pos})$ as described in Figure 8.

(d) If $((\mathsf{out}.c_1 = \perp) \vee (\mathsf{out}.c_2 = \perp))$, do the following.

   i. Let $p \in [k]$ be such that $\mathbf{x}_p.\mathsf{ind} = 1$ and denote $\mathsf{K}_0 = \mathbf{x}_p.\mathsf{K}$ as the root key for cPRF.

   ii. Derive the randomness for encryption at time step pos as $r_{\mathsf{pos}} = \mathsf{F.Eval}(\mathsf{K}_0, (\mathsf{pos} \| \mathsf{rand} \| \mathsf{gsalt}))$.

   iii. **Computing Encrypted Symbols using randomness derived from** cPRF. If $\mathsf{out}.c_1 = \perp$, do the following.
      - Compute the delegated PRF key $\mathsf{K}_{\mathsf{pos}+1} = \mathsf{F.KeyDel}(\mathsf{K}_0, f_{\mathsf{pos}+1})$. Set $\mathsf{key\text{-}id} = \mathsf{rand}$.
      - Compute the 1FE symbol ciphertext encoding $\sigma = \mathsf{inp}.u_1$ as $\mathsf{CT}_{\mathsf{sym},\mathsf{pos}} = \bar{\mathcal{E}}(\mathsf{1FE.PK}_1, \mathbf{y}_1; r_{\mathsf{pos}})$, where $\mathbf{y}_1 = (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{\mathsf{pos}+1}, \mathsf{pos}, \ell, \mathsf{gsalt}, \sigma, \mathsf{Trap})$.

   iv. **Computing Encrypted State for First Time Step.** If $((\mathsf{out}.c_2 = \perp) \wedge (\mathsf{pos} = 1))$, do the following.
      - Compute the 1FE state ciphertext encoding $\mathsf{q_{st}} = \mathsf{inp}.u_2$ as $\mathsf{CT}_{\mathsf{st},1} = \bar{\mathcal{E}}(\mathsf{1FE.PK}_2, \mathbf{y}_2; r_1)$, where $\mathbf{y}_2 = (\mathsf{ST}, \mathsf{q_{st}}, \mathsf{gsalt})$. Set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},1}$.

(e) If $\mathsf{pos} = 1$, output $\mathsf{out} = (\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$. Otherwise, output $\mathsf{out} = (\mathsf{CT}_{\mathsf{sym},\mathsf{pos}}, \perp)$.

Fig. 7. This circuit aggregates and re-randomizes the ciphertexts provided during encryption to use randomness derived from a cPRF. The seed for the cPRF is specified in the ciphertext for first party and the input is specified by the key. This ensures that each ciphertext, key pair form a unique "thread" of execution.

- **Aggregate the ciphertexts of all users.**

3. For $i = 1$ to k, do the following:
   (a) For $j = 1$ to ℓ_i, do the following:
      i. If $((i = 1) \wedge (j = 1))$, invoke $\mathsf{kFE.Dec}(\mathsf{SK}_{\mathsf{Agg}}, (\mathsf{CT}_{1,\mathsf{SYM},1}, \{\mathsf{CT}_{n,\mathsf{SP}}\}_{n \in [k] \setminus \{1\}}))$ to obtain $(\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$.
      ii. If $((i = 1) \wedge (j > 1))$, invoke $\mathsf{kFE.Dec}(\mathsf{SK}_{\mathsf{Agg}}, (\mathsf{CT}_{1,\mathsf{SYM},j}, \{\mathsf{CT}_{n,\mathsf{SP}}\}_{n \in [k] \setminus \{1\}}))$ to obtain $(\mathsf{CT}_{\mathsf{sym},j}, \perp)$.
      iii. Else, invoke $\mathsf{kFE.Dec}(\mathsf{SK}_{\mathsf{Agg}}, (\mathsf{CT}_{i,\mathsf{SYM},j}, \{\mathsf{CT}_{n,\mathsf{SP}}\}_{n \in [k] \setminus \{i\}}))$ to obtain $(\mathsf{CT}_{\mathsf{sym},\widetilde{L_i}+j}, \perp)$, where $\widetilde{L_i} = \sum_{m=1}^{i-1} \ell_m$.

- **Execute the TM on aggregated input.**

4. The aggregated sequence of ciphertexts under the Next scheme, of length $L_k = \sum_{j=1}^{k} \ell_j$ computed above is expressed as:
$((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \mathsf{CT}_{\mathsf{sym},2}, \ldots, \mathsf{CT}_{\mathsf{sym},\ell_1}, \mathsf{CT}_{\mathsf{sym},\ell_1+1}, \ldots, \mathsf{CT}_{\mathsf{sym},L_k})$.
5. Let $t = 1$. While the Turing machine does not halt, do:
   (a) Invoke $\mathsf{1FE.Dec}(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},t}, \mathsf{CT}_{\mathsf{st},t}))$ to obtain:
      – ACC or REJ. In this case, output "Accept" or "Reject" respectively, and exit the loop.
      – $(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1})$.
   Note that t' is the next time step that the work tape cell accessed at time step t will be accessed again.
   (b) Let $t = t + 1$ and go to start of loop.

```
Subroutine Trap-Mode_Agg(Trap, inp, rand, gsalt, $\ell, \mathcal{C}_1, \mathcal{C}_2$, pos)
Interpret inp $= (u_1, u_2) = (w_i, \mathsf{q}_{\mathsf{st}})$ and initialize out $= (c_1, c_2)$, where $c_1 = c_2 = \perp$.

If Trap.key-id = rand, do the following.
(a) If $((\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_3 = 1))$, do the following:
    i. If $((\mathsf{Trap.Sym\ TS} = \mathsf{pos}) \wedge (\mathsf{pos} \leq \ell))$, compute $\mathsf{CT}_{\mathsf{sym,pos}} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_1)$ and set out.$c_1 = \mathsf{CT}_{\mathsf{sym,pos}}$.
    ii. If $((\mathsf{Trap.ST\ TS} = \mathsf{pos}) \wedge (\mathsf{pos} = 1))$, compute $\mathsf{CT}_{\mathsf{st,pos}} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set out.$c_2 = \mathsf{CT}_{\mathsf{st},1}$.
(b) If $((\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_1 = 1))$, do the following:
    i. If $((\mathsf{Trap.Sym\ TS}_1 = \mathsf{pos}) \wedge (\mathsf{pos} \leq \ell))$, set inp.$u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol to be encrypted and output at time step pos.
    ii. If $((\mathsf{Trap.ST\ TS}_1 = \mathsf{pos}) \wedge (\mathsf{pos} = 1))$, set inp.$u_2 = \mathsf{Trap.ST\ val}_1$ with the start state to be encrypted and output at time step 1.
(c) If $((\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_2 = 1))$, do the following:
    i. If $((\mathsf{Trap.Sym\ TS}_2 = \mathsf{pos}) \wedge (\mathsf{pos} \leq \ell))$, set inp.$u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol to be encrypted and output at time step pos.
    ii. If $((\mathsf{Trap.ST\ TS}_2 = \mathsf{pos}) \wedge (\mathsf{pos} = 1))$, set inp.$u_2 = \mathsf{Trap.ST\ val}_2$ with the start state to be encrypted and output at time step 1.
(d) If Trap.global-salt < gsalt, set $b = 0$, if Trap.global-salt > gsalt, set $b = 1$.
    i. If pos $\neq 1$, update inp $= (\mathsf{Trap.val}_b, \perp)$; else update inp $= (\mathsf{Trap.val}_b, \mathsf{q}_{\mathsf{st}})$.

If Trap.key-id > rand, set $b = 1$, if Trap.key-id < rand set $b = 0$.
(a) If pos $\neq 1$, update inp $= (\mathsf{Trap.val}_b, \perp)$; else update inp $= (\mathsf{Trap.val}_b, \mathsf{q}_{\mathsf{st}})$.

Output. Return (inp, out).
```

Fig. 8. Subroutine handling the trapdoor modes in Agg. This is "active" only in the proof.

4.2 Correctness of Multi-Input TMFE

The proof of correctness is split into two parts. In the first part we argue that, given as input the secret key $\mathsf{SK}_{\mathsf{Agg}}$ along with k ciphertexts under the kFE scheme, exactly one of which encodes a symbol and the other $(k-1)$ encode the individual input lengths, the kFE.Dec algorithm computes a 1FE ciphertext component of the symbol with its updated position in the global string. By repeating this process for all symbols encoded by all users, we obtain a sequence of 1FE ciphertext components, each containing its updated position in the aggregated string. Additionally, each of these ciphertext components contains a global/aggregate salt that is generated from concatenating each individual encryptor's randomly generated salts. This global salt identifies the particular input combination being aggregated.

Correctness of the second part corresponds to the correct execution of the Turing machine on the aggregate sequence of ciphertexts, and this is exactly the same as in Section 3. As before, we maintain the invariant that at any time step t, the input to the 1FE.Dec algorithm is a complete 1FE ciphertext decomposed into two components corresponding to symbol and state (along with additional auxiliary inputs), both computed with the same randomness $\mathsf{F.Eval}(\mathsf{K}_0, (\mathsf{t} \| \mathsf{rand} \| \mathsf{gsalt}))$.

In more detail, we have the following. *Correctness of Aggregation.* Formally, let there be k users so that k ciphertexts $\{\mathsf{CT}_{\mathbf{w}_{\mathsf{ind}}}\}_{\mathsf{ind} \in [k]}$ are given as input to kTMFE.Dec algorithm. For all $\mathsf{ind} \in [k]$, let ℓ_{ind} be the length of input string of user ind. Each ciphertext $\mathsf{CT}_{\mathbf{w}_{\mathsf{ind}}}$ is a sequence $(\mathsf{CT}_{\mathsf{ind,SP}}, (\mathsf{CT}_{\mathsf{ind,SYM},1}, \ldots, \mathsf{CT}_{\mathsf{ind,SYM},\ell_{\mathsf{ind}}}))$ of ciphertexts, where the first component $\mathsf{CT}_{\mathsf{ind,SP}}$ encodes the input string length of user ind and the second component $\{\mathsf{CT}_{\mathsf{ind,SYM},i}\}_{i \in [\ell_{\mathsf{ind}}]}$ encodes in order the i-th symbol w_i of the actual input string $\mathbf{w}_{\mathsf{ind}} = (w_1, w_2, \ldots, w_{\ell_{\mathsf{ind}}})$ of the same user. These ciphertexts are generated under the kFE scheme with the master secret key kFE.MSK which supports a k-input functionality $\mathsf{Agg} := \mathsf{Agg}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{q}_{\mathsf{st}},\perp,\perp}$. Therefore, given secret key $\mathsf{SK}_{\mathsf{Agg}}$, we have:

Function $\mathsf{Next}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{M},\mathcal{C}_1,\mathcal{C}_2}\big((\mathbf{z}_1,\mathbf{z}_2)\big)$

(a) **Reading Current (Symbol, State) Pair and Looking up Transition Table.**
   i. Interpret $\mathbf{z}_1 = (\mathsf{type}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, \mathsf{gsalt}, s, \mathsf{Trap})$, $\mathbf{z}_2 = (\mathsf{type}, s, \mathsf{gsalt})$. If $((\mathbf{z}_1.\mathsf{type} \neq \mathsf{SYM}) \vee (\mathbf{z}_2.\mathsf{type} \neq \mathsf{ST}) \vee (\mathbf{z}_1.\mathsf{key\text{-}id} \neq \mathsf{rand}) \vee \wedge(\mathbf{z}_1.\mathsf{gsalt} \neq \mathbf{z}_2.\mathsf{gsalt}))$, output $\perp$ and abort.
   ii. Interpret $(\mathbf{z}_1.s, \mathbf{z}_2.s) = (\sigma_t, q_t)$ as the symbol, state pair for the current time step $\mathbf{z}_1.t = t$, input $\mathbf{z}_1.\mathsf{K}_{t+1} = \mathsf{K}_{t+1}$ as the constrained PRF key for future time steps. Denote $\mathsf{key\text{-}id} = \mathbf{z}_1.\mathsf{key\text{-}id}, \ell = \mathbf{z}_1.\ell, \mathsf{gsalt} = \mathbf{z}_1.\mathsf{gsalt}$ and $\mathsf{Trap} = \mathbf{z}_1.\mathsf{Trap}$. Using the transition table of the machine M, look up the next state q_{t+1} as well as the symbol $\sigma_{t'}$ to be written on the work-tape, where t' is the time step the current work tape cell will next be read by M. If q_{t+1} is an accept or reject state, then output ACC or REJ and exit.
   iii. Initialize $\mathsf{inp} = (\sigma_{t'}, \mathsf{q}_{t+1})$.

(b) **Choosing Real or Trapdoor mode.** If $\mathsf{Trap.mode\text{-}real} = 1$, initialize an output vector $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. Else invoke $\mathsf{Trap\text{-}Mode}_{\mathsf{Next}}\big(\mathsf{Trap}, \mathsf{inp}, \mathsf{rand}, \mathsf{gsalt}, \ell, \mathcal{C}_1, \mathcal{C}_2, t, t'\big)$ as described in Figure 10 to obtain $\big(\mathsf{inp} = (u_1, u_2), \mathsf{out} = (c_1, c_2)\big)$.

(c) **Computing Next Encrypted Symbol.** If $\mathsf{out}.c_1 = \perp$, do the following.
   i. Noting that $t' > t$, derive the randomness at time step t' using the delegated key K_{t+1} as $r_{t'} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t' \| \mathsf{rand} \| \mathsf{gsalt}))$. Compute the delegated PRF key $\mathsf{K}_{t'+1} = \mathsf{F.KeyDel}(\mathsf{K}_{t+1}, f_{t'+1})$.
   ii. Compute the 1FE ciphertext component encoding the symbol $\sigma_{t'} = \mathsf{inp}.u_1$ for time step t' as
   $$\mathsf{CT}_{\mathsf{sym},t'} = \bar{\mathcal{E}}\big(\mathsf{1FE.PK}_1, (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t'+1}, t', \ell, \mathsf{gsalt}, \sigma_{t'}, \mathsf{Trap}); r_{t'}\big)$$
   iii. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.

(d) **Computing Next Encrypted State.** If $\mathsf{out}.c_2 = \perp$, do the following.
   i. Derive the randomness at time step $t+1$ as $r_{t+1} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t+1 \| \mathsf{rand} \| \mathsf{salt}))$ and compute the $\mathsf{1FE}_2$ ciphertext component encoding the state $\mathsf{q}_{t+1} = \mathsf{inp}.u_2$ for time step $t+1$ as
   $$\mathsf{CT}_{\mathsf{st},t+1} = \bar{\mathcal{E}}\big(\mathsf{1FE}_2.\mathsf{PK}_2, (\mathsf{ST}, \mathsf{q}_{t+1}, \mathsf{gsalt}); r_{t+1}\big)$$
   ii. Set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},t+1}$.

(e) **Output :** $\mathsf{out} = \big(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1}\big)$

Fig. 9. Function to mimic TM computation. It reads the current symbol, state pair and outputs an encryption of the new state and symbol to be written under the appropriate randomness generated using a cPRF.

Subroutine $\mathsf{Trap\text{-}Mode}_{\mathsf{Next}}\big(\mathsf{Trap}, \mathsf{inp}, \mathsf{rand}, \mathsf{gsalt}, \ell, \mathcal{C}_1, \mathcal{C}_2, t, t'\big)$

Interpret the input vector $\mathsf{inp} = (u_1, u_2) = (\sigma_{t'}, \mathsf{q}_{t+1})$ and initialize the output vector $\mathsf{out} = (c_2, c_2)$, where $c_1 = c_2 = \perp$.

1. If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_3 = 1)\big)$, do the following.
   (a) If $\big((\mathsf{Trap.Sym\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t') \wedge (t > \ell)\big)$, compute the $\mathsf{1FE}_2$ symbol ciphertext $\mathsf{CT}_{\mathsf{sym},t'} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_1)$ and set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.
   (b) If $\big((\mathsf{Trap.ST\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t+1) \wedge (t > 1)\big)$, compute the $\mathsf{1FE}_2$ state ciphertext $\mathsf{CT}_{\mathsf{st},t+1} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},t+1}$.
2. If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_1 = 1)\big)$, do the following:
   (a) If $\big((\mathsf{Trap.Sym\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_1$ to be encrypted and given as output for time step t'.
   (b) If $\big((\mathsf{Trap.ST\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_1$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_1$ to be encrypted and given as output for time step $t+1$.
3. If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_2 = 1)\big)$, do the following:
   (a) If $\big((\mathsf{Trap.Sym\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_2$ to be encrypted and given as output for time step t'.
   (b) If $\big((\mathsf{Trap.ST\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_2$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_2$ to be encrypted and given as output for time step $t+1$.
4. Exit the subroutine returning $(\mathsf{inp}, \mathsf{out})$.

Fig. 10. Subroutine handling the trapdoor modes in Next. This is "active" only in the proof.

1. Invoking $\mathsf{kFE.Dec}$ on the ciphertext $\mathsf{CT}_{1,\mathsf{SYM},1}$ encoding the first symbol of $\mathbf{w}_1$ along with the special ciphertexts $\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}$ encoding $|\mathbf{w}_{\mathsf{ind}}|$ for ind $\neq 1$ gives $(\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$. By correctness of kFE decryption, we have: $\mathsf{kFE.Dec}\left(\mathsf{SK}_{\mathsf{Agg}}, \left(\mathsf{CT}_{1,\mathsf{SYM},1}, \{\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}\}_{\mathsf{ind}\in[k]\setminus\{1\}}\right)\right) = (\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$.
2. Invoking $\mathsf{kFE.Dec}$ on the ciphertext $\mathsf{CT}_{1,\mathsf{SYM},j}$ encoding the jth symbol of $\mathbf{w}_1$ along with the special ciphertexts $\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}$ encoding $|\mathbf{w}_{\mathsf{ind}}|$ for ind $\neq 1$ gives $(\mathsf{CT}_{\mathsf{sym},j}, \perp)$. By correctness of kFE decryption, we have: $\mathsf{kFE.Dec}\left(\mathsf{SK}_{\mathsf{Agg}}, \left(\mathsf{CT}_{1,\mathsf{SYM},j}, \{\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}\}_{\mathsf{ind}\in[k]\setminus\{1\}}\right)\right) = (\mathsf{CT}_{\mathsf{sym},j}, \perp)$.
3. Finally, $\forall$ ind $\in [k] \setminus \{1\}$, invoking $\mathsf{kFE.Dec}$ on the ciphertext $\mathsf{CT}_{\mathsf{ind},\mathsf{SYM},j}$ encoding the jth symbol of $\mathbf{w}_{\mathsf{ind}}$ along with the special ciphertexts $\mathsf{CT}_{\mathsf{ind}',\mathsf{SP}}$ encoding $|\mathbf{w}_{\mathsf{ind}'}|$ for ind $\neq$ ind$'$ computes the new global position of the symbol in the aggregated string and outputs $\left(\mathsf{CT}_{\mathsf{sym},\widetilde{L_i}+j}, \perp\right)$. By correctness of kFE decryption, we have: $\mathsf{kFE.Dec}\left(\mathsf{SK}_{\mathsf{Agg}}, \left(\mathsf{CT}_{\mathsf{ind},\mathsf{SYM},j}, \{\mathsf{CT}_{\mathsf{ind}',\mathsf{SP}}\}_{\mathsf{ind}'\in[k]\setminus\{\mathsf{ind}\}}\right)\right) = \left(\mathsf{CT}_{\mathsf{sym},\widetilde{L_i}+j}, \perp\right)$, where $\widetilde{L_i} = \sum_{m=1}^{\mathsf{ind}-1} \ell_m$.

Note that $\mathsf{F.Eval}(\mathsf{K}_0, (\mathsf{pos}\|\mathsf{rand}\|\mathsf{gsalt}))$ is the randomness used to compute each of these ciphertext components, where pos refers to the global position specific to a symbol in the aggregate input string.

Correctness of TM Execution. The 1FE scheme supports the functionality $\mathsf{Next} := \mathsf{Next}_{\mathsf{1FE.PK},\mathsf{rand},M,\perp,\perp}$. Let the newly generated and organized sequence of ciphertexts based on time steps be as follows: $\left((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \{\mathsf{CT}_{\mathsf{sym},i}\}_{i\in[2,L_k]}\right)$ with $L_k = \sum_{i=1}^{k} \ell_i$. Let $\mathbf{w} = (w_1, w_2, \ldots, w_{\ell_1}, w_{\ell_1+1}, w_{\ell_1+2}, \ldots, w_{\ell_1+\ell_2}, \ldots, w_{L_k})$ be the aggregated input string and define $\tau = \mathsf{runtime}(M, \mathbf{w})$. For any time step $t \in [\tau - 2]$, we have

1. Let $t \in [\tau - 2] \setminus [\ell]$. If the current work tape cell was accessed[12], at some time step $\tilde{t} < t$, then $\mathsf{CT}_{\mathsf{sym},t}$ encoding $(\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, \mathsf{gsalt}, \sigma_t, \mathsf{Trap})$ was constructed at time step $\tilde{t}$. Note that σ_t may be the blank symbol β. When $t \in [\ell]$, $\mathsf{CT}_{\mathsf{sym},t}$ is constructed at time step t via the Agg circuit.
2. The ciphertext component $\mathsf{CT}_{\mathsf{st},t}$ encoding $(\mathsf{ST}, \mathsf{q}_t, \mathsf{gsalt})$ at time step t was constructed at time step $t-1$ for $t > 1$ and at time step 1, when $t = 1$.
3. The randomness $r_t = \mathsf{F.Eval}(\mathsf{K}_{\tilde{t}+1}, (\mathsf{t}\|\mathsf{rand}\|\mathsf{gsalt})) = \mathsf{F.Eval}(\mathsf{K}_{\mathsf{t}}, (\mathsf{t}\|\mathsf{rand}\|\mathsf{gsalt}))$ binds $\mathsf{CT}_{\mathsf{sym},t}$ and $\mathsf{CT}_{\mathsf{st},t}$ and both the encoded messages also share the same global salt.

Thus, at any given time step $t \in [\tau - 2]$, we have a complete ciphertext of 1FE which may be fed again with $\mathsf{SK}_{\mathsf{Next}}$ to 1FE.Dec in order to proceed with the computation. Thus, the execution of 1FE.Dec at the $(\tau - 2)^{\text{th}}$ time step provides the complete pair $(\mathsf{CT}_{\mathsf{sym},\tau-1}, \mathsf{CT}_{\mathsf{st},\tau-1})$. By the correctness of 1FE scheme again, at time step $t = \tau - 1$, invoking $\mathsf{1FE.Dec}(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},\tau-1}, \mathsf{CT}_{\mathsf{st},\tau-1}))$ outputs either “Accept” or “Reject” by simulating the execution of M for the final time step τ inside the function Next, thus correctly outputting $M(\mathbf{w})$.

4.3 Proof of Security for multi-input TMFE

Security of the above construction follows the same blueprint as the proof in Section 3 except that instead of single input functionality ReRand, we now use a k-input functionality Agg to aggregate and rerandomize the inputs. We emphasize that the outputs produced by the Agg functionality are *exactly the same* as the outputs produced by ReRand functionality in Section 3: namely a sequence of 1FE ciphertexts encoding the symbol and global position, computed using randomness derived from a cPRF. Hence, the chief new ingredient in the security proof is the security of Agg functionality, which is derived from the security of the kFE scheme.

Formally, we argue that:

Theorem 2. *Assume that the k input FE for circuits* kFE *satisfies standard indistinguishability, and the single input FE for circuits* 1FE *satisfies distributional indistinguishability. Assume that the cPRF*

[12] We assume that every time a cell is accessed, it is written to, by writing the same symbol again if no change is made.

is secure according to definition. Then, the above construction of k input kTMFE *satisfies standard indistinguishability.*

The proof follows the outline of the single input case, except that now we must additionally keep track of multiple execution threads corresponding to various combinations of ciphertexts across multiple users, i.e. various "global salt" values. In more detail, if each of k users makes Q ciphertext requests, then we have Q^k total possible combinations of ciphertexts, each yielding a different execution thread per key. Note that each of the Q^k combinations is identified with a unique "global salt". We will assume w.l.o.g that there is a lexicographic ordering on all the global salt values; this can be easily ensured by associating a counter value with each random salt. We do not explicitly include this for notational brevity.

In the single input case, we replaced the execution chain of a machine over an input string from $b = 0$ to $b = 1$, step by step, and enumerated over all keys. Now, we again replace an execution chain step by step as in the single input case, but additionally enumerate over all Q^k combinations for each key, as well as over all keys as before. The number of hybrids grows multiplicatively by Q^k. Details are again deferred to in the full version [1].

5 Indistinguishability Obfuscation for Turing Machines

In this section we construct indistinguishability obfuscation for Turing machines with bounded length input, i.e. the input length $n = n(\lambda)$ is any fixed polynomial in the security parameter. To support inputs of length n, we need an $(n+1)$-ary miFE for Turing machines denoted as (n+1)-TMFE; we instantiate this with our construction from Section 4.

5.1 Construction

Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ denote an ensemble of Turing machines with alphabet $\Sigma_\lambda = \{0,1\}$. Let $\mathsf{Encode} = \{\mathsf{Encode}_\lambda : \mathcal{M}_\lambda \to \Sigma_{\mathsf{enc}}^*\}_{\lambda\in\mathbb{N}}$ be an ensemble of encoding schemes for $\mathcal{M}$ on alphabet Σ_{enc} such that for any $M \in \mathcal{M}_\lambda$, $\mathsf{Encode}_\lambda(M) = \langle M\rangle$. Further, let $\mathcal{U} = \{\mathsf{U}_\lambda\}_{\lambda\in\mathbb{N}}$ denote the set of Universal Turing machines parameterized by the security parameter with alphabet $\Sigma_{\mathcal{U}} = \Sigma_{\mathsf{enc}} \cup \Sigma_\lambda$ such that for all $\lambda \in \mathbb{N}$, for any $M \in \mathcal{M}_\lambda$ and any $\mathbf{x} = (x_1, \ldots, x_n) \in \Sigma_\lambda^n$, $\mathsf{U}_\lambda(\mathbf{x}, \langle M\rangle)$ takes $\mathbf{x}$ and an encoding $\langle M\rangle$ of M, simulates M on $\mathbf{x}$ and outputs $M(\mathbf{x})$.

Let (n+1)-TMFE denote the $(n+1)$-ary multi-input functional encryption scheme for Turing machines with alphabet $\Sigma_{\mathcal{U}}$. We construct an ensemble of indistinguishability obfuscators $\mathsf{iO} = \{\mathsf{iO}_\lambda\}_{\lambda\in\mathbb{N}}$ with $\mathsf{iO}_\lambda = (\mathsf{iO.Obf}, \mathsf{iO.Eval})$ for $\mathcal{M}_\lambda$ with inputs $\mathbf{x} \in \Sigma_\lambda^n$ as follows.

$\mathsf{iO.Obf}(1^\lambda, 1^n, M)$**:** On input the security parameter λ, a bound $n \in \mathbb{N}$ and a Turing machine $M \in \mathcal{M}_\lambda$, do the following:

1. Compute the encoding of M as $\mathsf{Encode}_\lambda(M) = \langle M\rangle$.
2. Compute a master secret key $\mathsf{MSK} \leftarrow \mathsf{(n+1)\text{-}TMFE.Setup}(1^\lambda, 1^{n+1})$.
3. Compute the secret key for machine U_λ as $\mathsf{SK_U} \leftarrow \mathsf{(n+1)\text{-}TMFE.KeyGen}(\mathsf{MSK}, \mathsf{U}_\lambda)$.
4. For $i \in [n]$, compute the encryptions $\mathsf{CT}_i^b = \mathsf{(n+1)\text{-}TMFE.Enc}(\mathsf{MSK}, (\mathsf{b}, \mathsf{i})), \mathsf{b} \in \Sigma_\lambda$.
5. Compute the encoding of M as $\mathsf{CT}_{n+1} = \mathsf{(n+1)\text{-}TMFE.Enc}(\mathsf{MSK}, (\langle\mathsf{M}\rangle, \mathsf{n}+1))$.
6. Output the obfuscated machine as $\widetilde{M} = \big(\mathsf{SK_U}, \big(\{\mathsf{CT}_i^b\}_{i\in\{1,\ldots,n\}, b\in\Sigma_\lambda}, \mathsf{CT}_{n+1}\big)\big)$.

$\mathsf{iO.Eval}(\widetilde{M}, \mathbf{x})$**:** On input the obfuscated machine $\widetilde{M}$ and an input $\mathbf{x} \in \Sigma_\lambda^n$, do the following:

1. Parse $\widetilde{M} = \big(\mathsf{SK_U}, \big(\{\mathsf{CT}_i^b\}_{i\in\{1,\ldots,n\}, b\in\Sigma_\lambda}, \mathsf{CT}_{n+1}\big)\big)$ and $\mathbf{x} = (x_1, \ldots, x_n)$.
2. Compute and output $\mathsf{(n+1)\text{-}TMFE.Dec}\left(\mathsf{SK_U}, (\mathsf{CT}_1^{x_1}, \ldots, \mathsf{CT}_n^{x_n}, \mathsf{CT}_{n+1})\right)$.

Correctness is directly followed by the correctness of (n+1)-TMFE scheme. Since the (n+1)-TMFE we use is compact, the obfuscation size obtained by the above scheme is $\mathsf{poly}(\lambda, |\mathsf{U}|, |M|, n)$. In the full version [1], we show that our construction is secure:

Theorem 3. *Assume that* $(\mathsf{n}+1)\text{-}\mathsf{TMFE}$ *is a* 1*-key,* 2*-ciphertext selectively secure* $(n+1)$*-ary multi-input functional encryption scheme for Turing machines which satisfies standard indistinguishability. Then the construction in Section 5.1 is a secure indistinguishability obfuscator for the Turing machines with bounded input length* n.

Acknowledgement. We thank Vinod Vaikuntanathan for suggesting the generic transformation from FE to decomposable FE.

References

1. Shweta Agrawal and Monosij Maitra. Fe and io for turing machines from minimal assumptions. Cryptology ePrint Archive, Report 2018/, 2018. `http://www.cse.iitm.ac.in/~shwetaag/research/tm-mife-full.pdf`.
2. Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from lwe. In *ICALP*, 2017.
3. Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *Crypto*, 2015.
4. Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating ram computations with adaptive soundness and privacy. In *Theory of Cryptography Conference*, pages 3–30. Springer, 2016.
5. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, pages 308–326. Springer, 2015.
6. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In *Crypto*. Springer, 2017.
7. Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography Conference (TCC)*, pages 125–153. Springer, 2016.
8. Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In *EUROCRYPT*, 2017.
9. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.
10. Saikrishna Badrinarayanan, Divya Gupta, Abhishek Jain, and Amit Sahai. Multi-input functional encryption for unbounded arity functions. In *Advances in Cryptology–ASIACRYPT 2015*, pages 27–51. Springer, 2015.
11. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.
12. Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.
13. Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. In *Theory of Cryptography Conference*, pages 391–418. Springer, 2016.
14. Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1480–1498. IEEE, 2015.
15. Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.
16. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.
17. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Asiacrypt*. Springer, 2013.
18. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public Key Cryptography*, pages 501–519, 2014.
19. Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In *Eurocrypt*, 2016.
20. Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Succinct adaptive garbled ram. Cryptology ePrint Archive, Report 2015/1074, 2015. `https://eprint.iacr.org/2015/1074`.
21. Ran Canetti and Justin Holmgren. Fully succinct garbled ram. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 169–178. ACM, 2016.
22. Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and ram programs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, 2015.
23. Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC*, 2015.
24. Brent Carmer, Alex J Malozemoff, and Mariana Raykova. 5gen-c: multi-input functional encryption and program obfuscation for arithmetic circuits. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 747–764. ACM, 2017.
25. 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, 2015.

26. Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1115–1127. ACM, 2016.
27. 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. http://eprint.iacr.org/.
28. Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In *CRYPTO*, pages 579–604. Springer, 2016.
29. Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. Technical report, Cryptology ePrint Archive, Report 2016/102, 2016. http://eprint. iacr. org/2016/102, 2016.
30. Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In *Theory of Cryptography Conference*, pages 419–442. Springer, 2016.
31. Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2014.
32. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.
33. Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, pages 578–602, 2014.
34. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO (2)*, pages 536–553, 2013.
35. Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In *Theory of Cryptography Conference*, pages 40–71. Springer, 2017.
36. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, 2013.
37. Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Indistinguishability obfuscation for all circuits from secret-key functional encryption. *IACR Cryptology ePrint Archive*, 2017:361, 2017.
38. Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 603–648. Springer, 2018.
39. Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Simple and generic constructions of succinct functional encryption. In *IACR International Workshop on Public Key Cryptography*, pages 187–217. Springer, 2018.
40. Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. In *EUROCRYPT*, 2017.
41. Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, 2015.
42. Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In *Theory of Cryptography Conference*, pages 443–468. Springer, 2016.
43. Huijia Lin. Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In *Crypto*, 2017.
44. Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. In *TCC-A*, 2016.
45. Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local prgs. In *Crypto*, 2017.
46. Qipeng Liu and Mark Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In *Theory of Cryptography Conference*, pages 138–169. Springer, 2017.
47. Steve Lu and Rafail Ostrovsky. How to garble ram programs? In *Advances in Cryptology – EUROCRYPT*, 2013.
48. Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.
49. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
50. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *STOC*, 2014. http://eprint.iacr.org/2013/454.pdf.

A Construction: Decomposable FE for Circuits

Given any single-input circuit FE scheme 1FE satisfying standard indistinguishability based security, a *projective* garbled circuit scheme $\mathsf{GC} = (\mathsf{GCirc}, \mathsf{GInp}, \mathsf{GEval})$ with indistinguishability based security [35] supporting a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ with n-bit inputs, a simple PRF $\mathsf{F} = (\mathsf{F.Setup}, \mathsf{F.Eval})$ and a symmetric encryption scheme SYM, we can construct a single-input decomposable FE scheme DFE supporting the circuit class $\mathcal{C}$. We note that projective garbled circuit schemes satisfying indistinguishability based security are implied from one-way functions [35].

$\mathsf{DFE.Setup}(1^\lambda, 1^n)$: On input the security parameter λ and input message size n, do the following:

1. Generate $(\mathsf{1FE.PK}, \mathsf{1FE.MSK}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, 1^{2\lambda+\log n+2})$.
2. Output $(\mathsf{PK}, \mathsf{MSK}) = (\mathsf{1FE.PK}, \mathsf{1FE.MSK})$.

$\mathsf{DFE.Enc}(\mathsf{PK}, \mathbf{x})$**:** On input the public key PK and a message $\mathbf{x} = (x_1, \ldots, x_n)$ of length $n = |\mathbf{x}|$, do the following:

1. Sample a PRF key $\mathsf{K} \leftarrow \mathsf{F.Setup}(1^\lambda)$ and set a flag $\mathsf{mode} = 0$.
2. Compute $\mathsf{CT}_{x_i} = \mathsf{1FE.Enc}(\mathsf{PK}, (\mathsf{K}, \mathbf{0}, i, x_i, \mathsf{mode})), \forall i \in [n]$ and output $\mathsf{CT}_\mathbf{x} = \{\mathsf{CT}_{x_i}\}_{i\in[n]}$.

$\mathsf{DFE.KeyGen}(\mathsf{MSK}, C)$**:** On input the master secret key MSK and a circuit $C \in \mathcal{C}_\lambda$, do the following:

1. Sample a random $\mathsf{salt} \leftarrow \{0,1\}^\lambda$, $\mathsf{CT}_i \leftarrow \{0,1\}^{\ell(\lambda)}, \forall i \in [0, n]$.
2. Output $\mathsf{SK}_{\widehat{C}} = \mathsf{1FE.KeyGen}(\mathsf{MSK}, \widehat{C}_{C,\mathsf{salt},\{\mathsf{CT}_i\}_{i\in[n]},\mathsf{CT}_0})$, where $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$ is a circuit described in Figure 11.

Functionality $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}(\mathsf{K}, \mathsf{SYM.K}, i, x_i, \mathsf{mode})$

```
(a) Initialize the vector out = $(c_1, c_2)$, where $c_j = \perp, \forall j \in [2]$.
(b) If $\mathsf{mode} = 1$, do the following:
     i. Let $\mathsf{out}.c_1 = \mathsf{SYM.Dec}(\mathsf{SYM.K}, \mathsf{SYM.CT}_i)$.
    ii. If $i = 1$, let $\mathsf{out}.c_2 = \mathsf{SYM.Dec}(\mathsf{SYM.K}, \mathsf{SYM.CT}_{\widetilde{C}})$.
(c) If $\mathsf{mode} = 0$, do the following:
     i. Compute randomness $r = \mathsf{F.Eval}(\mathsf{K}, \mathsf{salt})$.
    ii. Use randomness $r$ to generate the garbled circuit for $C$ as $(\widetilde{C}, \mathsf{sk}) = \mathsf{GCirc}(1^\lambda, C; r)$ as well as to generate the label
        corresponding to the $i^{\text{th}}$ input wire as $\ell_{i,x_i} = \mathsf{GInp}(\mathsf{sk}, (x_i, i); r)$.
   iii. Let $\mathsf{out}.c_1 = \ell_{i,x_i}$. If $i = 1$, let $\mathsf{out}.c_2 = \widetilde{C}$.
(d) **Output :** out.
```

Fig. 11. Functionality $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$

$\mathsf{DFE.Dec}(\mathsf{SK}_{\widehat{C}}, \mathsf{CT}_\mathbf{x})$**:** On input a function key $\mathsf{SK}_{\widehat{C}}$ and a decomposed ciphertext $\mathsf{CT}_\mathbf{x} = \{\mathsf{CT}_{x_i}\}_{i\in[n]}$, do the following:

1. For $i = 1$, invoke $\mathsf{1FE.Dec}(\mathsf{SK}_{\widehat{C}}, \mathsf{CT}_{x_1})$ to obtain a pair $(\ell_{1,x_1}, \widetilde{C})$.
2. For all $i \in [2, n]$, invoke $\mathsf{1FE.Dec}(\mathsf{SK}_{\widehat{C}}, \mathsf{CT}_{x_i})$ to obtain $(\ell_{i,x_i}, \perp)$.
3. Note that $\widetilde{\mathbf{x}} = \{\ell_{i,x_i}\}_{i\in[n]}$ represents the labels corresponding to the garbled input underlying $\mathsf{CT}_\mathbf{x}$ generated as outputs of $\widehat{C}$, while $\widetilde{C}$ represents the garbled circuit for C.
4. Run $\mathsf{GEval}(\widetilde{C}, \widetilde{\mathbf{x}})$ to get $\mathbf{y}$.

Correctness. We have by correctness of 1FE.Dec that it outputs the garbled input $\widetilde{\mathbf{x}}$ and the garbled circuit $\widetilde{C}$ correctly. The correctness of GEval implies that decryption recovers $C(\mathbf{x})$ as desired.

The proof of security is provided in the full version [1].