Reusable Garbled Deterministic Finite Automata from Learning With Errors

Shweta Agrawal[1] and Ishaan Preet Singh[2]

1 **IIT Madras**
shweta@iitm.ac.in
2 **IIT Delhi**
ishaanps92@gmail.com

Abstract

We provide a single-key functional encryption scheme for Deterministic Finite Automata (DFA). The secret key of our scheme is associated with a DFA M, and a ciphertext is associated with an input $\mathbf{x}$ of *arbitrary* length. The decryptor learns $M(\mathbf{x})$ and nothing else. The ciphertext and key sizes achieved by our scheme are optimal – the size of the public parameters is independent of the size of the machine or data being encrypted, the secret key size depends only on the machine size and the ciphertext size depends only on the input size.

Our scheme achieves full functional encryption in the "private index model", namely the entire input $\mathbf{x}$ is hidden (as against $\mathbf{x}$ being public and a single bit b being hidden). Our single key FE scheme can be compiled with symmetric key encryption as in [18] to yield reusable garbled DFAs for arbitrary size inputs, that achieves machine and input privacy along with reusability under a strong simulation based definition of security.

We generalize this to a functional encryption scheme for Turing machines TMFE which has short public parameters that are independent of the size of the machine or the data being encrypted, short function keys, and input-specific decryption time. However, the ciphertext of our construction is large and depends on the worst case running time of the Turing machine (but not its description size). These provide the first FE schemes that support unbounded length inputs, allow succinct public and function keys and rely on LWE.

Our construction relies on a new and arguably natural notion of *decomposable functional encryption* which may be of independent interest.

1998 ACM Subject Classification E.3 Data Encryption

Keywords and phrases Functional Encryption, Learning With Errors, Deterministic Finite Automata, Garbled DFA

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Functional encryption permits *controlled disclosure* of encrypted data, enabling the evaluator to learn some authorised function of encrypted data in the clear. In functional encryption (FE), a secret key corresponds to a function f and ciphertexts correspond to strings from the domain of f. Given a function SK_f and a ciphertext $\mathsf{CT}_{\mathbf{x}}$, the decryptor learns $f(\mathbf{x})$ and nothing else. Functional encryption has found diverse applications, such as spam filtering on encrypted data [18], online dating [20], delegation of computation [24] and many others.

The function embedded within the secret key in FE is typically represented as a circuit. While circuits are a powerful model of computation, the circuit representation has significant drawbacks in practical scenarios. Consider the application of spam filtering on encrypted

emails, where the email gateway may be given a key to test the incoming email for spam. Representing the computation as a circuit forces emails to be of a fixed length – a requirement which is ill-fitting and wasteful. Another significant drawback of the circuit model is that it incurs worst case running time on every input.

In practice, most spam filters as well as malware and intrusion detection systems are implemented using pattern matching operations represented as deterministic finite automata (DFA) [28, 21, 6, 14]. Note that in all these applications, the size of the input is highly variable and instance specific, and restricting it to be of fixed length is cumbersome. Therefore a functional encryption scheme for DFAs which supports dynamic data length would be the "right fit" in such situations. However, although functional encryption for circuits has been constructed based on the hardness of Learning With Errors (LWE) in the single key setting, it is unclear how to leverage these techniques to support the arbitrary data length required by DFAs.

1.1 Our Results.

In this work, we provide a single-key functional encryption scheme for Deterministic Finite Automata (DFA). The secret key of our scheme is associated with a DFA M, and a ciphertext is associated with an input $\mathbf{x}$ of *arbitrary* length. The decryptor learns $M(\mathbf{x})$ and nothing else. The ciphertext and key sizes achieved by our scheme are optimal[1] – the public key size is independent of the machine and input size, the secret key size depends only on the machine size and the ciphertext size depends only on the input size.

Our scheme achieves full functional encryption in the "private index model", namely the entire input $\mathbf{x}$ is hidden (as against $\mathbf{x}$ being public and a single bit b being hidden). Our single key FE scheme can be compiled with symmetric key encryption as in [18] to yield reusable garbled DFAs for arbitrary size inputs, that achieves machine and input privacy along with reusability under a strong simulation based definition of security.

We generalize this to a functional encryption scheme for Turing machines TMFE which has short public parameters that are independent of the size of the machine or the data being encrypted, short function keys, and input-specific decryption time. However, the ciphertext of our construction is large and depends on the worst case running time of the Turing machine (but not its description size). These provide the first FE schemes that support unbounded length inputs, allow succinct public and function keys and rely on LWE.

Our construction relies on a new and arguably natural notion of *decomposable functional encryption* which may be of independent interest.

1.2 Related Work.

Functional encryption for DFAs has received some attention already. Closest to our work is the "Attribute Based Encryption" scheme for DFAs constructed by Waters [29]. In [29], the encrypt algorithm takes as input a pair $(\mathbf{x}, b)$ where $\mathbf{x}$ may be of arbitrary size, and b is a bit. The key corresponds to a DFA machine M so that given a key for M and a ciphertext for $(\mathbf{x}, b)$, the decryptor learns the bit b if and only if M accepts $\mathbf{x}$. Note that in contrast to our work, the vector $\mathbf{x}$ is not hidden by the construction, neither is the machine M; only the bit b is hidden. On the other hand, the construction [29] can support polynomially many keys, whereas ours can only support a single key. Attrapadung [5] extended the work of

[1] upto logarithmic factors.

Waters [29] to achieve adaptive rather than selective security. Another work that constructs Attribute Based Encryption for DFAs is by Boyen and Li [8]. However, in their construction, the input size to the DFA must be bounded in advance; avoiding this restriction is the main motivation for our work.

There are other known functional encryption systems that support unbounded size inputs, even supporting Turing machines, achieving input specific runtime and dynamic data length [17, 2, 7, 23, 12, 13]. However, the mildest assumption required by this line of work is the existence of indistinguishability obfuscation.

From standard assumptions, single key functional encryption has been constructed for all polynomial sized circuits [27, 18]. A natural approach to construct reusable garbled DFA/TM then, is to convert the machine to a circuit and leverage the constructions of [27, 18]. However, instantiating this compiler with the reusable garbled circuits construction [18] leads to a construction that cannot support dynamic data lengths, which is the main focus of this work. On the other hand, using the construction by Sahai and Seyalioglu [27] leads to a DFA/TM FE construction with large public key and ciphertext size, since the construction by [27] suffers from public key and ciphertext size that depend on the circuit size. Please see Appendix A for more details.

1.3 Our Techniques.

To begin, we describe our single key FE scheme for DFA. Next, we describe how this construction may be generalized to Turing machines.

1.3.1 Single Key FE for DFA.

We briefly recall how a DFA works. A DFA machine M is represented by the tuple $(Q, \Sigma, T, q_{st}, F)$ where Q is a finite set of states, Σ is a finite alphabet, $T : \Sigma \times Q \to Q$ is the transition function, q_{st} is the start state, $F \subseteq Q$ is the set of accepting states. Upon input $\mathbf{w} \in \Sigma^k$ for some arbitrary polynomial k, the machine M accepts $\mathbf{w}$ if and only if there exists a sequence of states $q_1, \ldots, q_k$ so that $q_1 = q_{st}$, $T(w_i, q_i) = q_{i+1}$ for $i \in [k-1]$, and $q_k \in F$.

To mimic the DFA computation, a natural starting point is to imagine a function key that stores the transition table of a DFA, receives as input the current (symbol, state) pair and produces as output an encryption of the next state of the computation. In more detail, say the encryptor provides encryptions of each input symbol x_i, for $i \in [|\mathbf{x}|]$, in addition to an encryption for the first (fixed) state q_{st}. Now, the function key could accept 2 inputs (x_1, q_{st}), lookup the transition table and produce an encryption of the next state q_2. Suppose this encryption can only be paired with the encryption of x_2 and none other, then we could provide (x_2, q_2) as input to the function in the next step, thus propagating the computation.

We tie together encryptions of symbol with encryptions of state via the notion of *decomposable functional encryption*. Intuitively, decomposability requires that the public key PK and the ciphertext $\mathsf{CT}_{\mathbf{y}}$ of a functional encryption scheme be decomposable into components PK_j and CT_j for $j \in [|\mathbf{y}|]$, where CT_j depends on a single deterministic bit y_j and the public key component PK_j. All components CT_j are tied together by *common randomness* used for their creation, although each CT_j may use additional independent randomness. Aside from the message dependent components, a ciphertext can contain components that are independent of the message and depend only on the common randomness. The main advantage offered by decomposable functional encryption is that given the common randomness, each ciphertext component CT_j can be constructed independently of the rest. These components can then be joined together to create a complete ciphertext which can then be decrypted

successfully. Additionally, only components that were constructed using the same randomness can be "joined", thereby preventing mix and match attacks where an adversary tries to treat mismatched symbol state pairs such (x_3, q_2) as a single legitimate input.

Now, suppose we have a decomposable functional encryption scheme for circuits. Then, we may proceed with the aforementioned strategy and divide the ciphertext into two components – the first encoding the current symbol, and the second encoding the current state. We may use the function key to generate the second component, *using the same common randomness that was used to generate the first component.*

To take this approach forward we must find a suitable decomposable functional encryption scheme for circuits – fortunately most functional encryption schemes in the literature are decomposable. In particular, we show that the the succinct single key FE by Goldwasser et al. [18] is decomposable. This scheme appears suitable for our purposes as the ciphertext and public key in this scheme are independent of circuit size.

However, note that the ciphertext of [18] suffers from *output-size dependence*, i.e. it grows linearly with the output length of the circuit. This implies that the function key may not produce an output that is proportional to the length of the ciphertext. To obtain a (single key) construction from LWE, we resolve this issue by repurposing a classic trick from Yao's garbled circuit construction, so that the output length of the circuit can be made independent of the ciphertext size, at the cost of blowing up the ciphertext size somewhat. More concretely, instead of having the circuit output a new ciphertext, the encryptor provides symmetric key encryptions of CktFE [18] ciphertext components, encrypting *all possible bit values* (nesting CktFE ciphertext inside SKE ciphertext), and the function key outputs the SKE keys to unlock the correct CktFE ciphertext components, corresponding to the bit values chosen by the key. This allows us to *select* the next state with a circuit output length independent of the ciphertext size. For more details, we refer the reader to Section 4. This provides input privacy and reusability but not machine privacy. We achieve machine privacy following ideas of [18] – see Appendix E for details.

1.3.2 Single key FE for Turing Machines.

To extend the above construction to support Turing Machines, we must address two challenges: a) head movements should not reveal anything about the input and b) we need to write to the tape. Below we describe how to handle each challenge in turn.

To overcome the first challenge, a natural approach is to use oblivious TMs, which fix the head movement of a TM to be independent of the input. Moreover, there exist efficient transformations that convert any Turing machine M that takes time T to decide an input to an oblivious one that takes time $T \log T$ to decide the same input [25]. It remains to address the challenge of handling tape writes. Since the head movements of the TM are now fixed, the only thing that the transition function must specify is the next state, and the symbol that must be written to the current tape cell. We leverage decomposability and have the encryptor provide a ciphertext component encoding state, and another component encoding current work tape symbol *for every step in the computation.* Indeed, this forces our ciphertext to depend (linearly) on worst-case runtime of the Turing machine. All the ciphertext components for a given time step are tied together with common randomness as before. To ensure that the decryptor only learns the ciphertext components corresponding to the particular state and tape symbol that occur during computation, the encryptor encrypts all CktFE ciphertexts with symmetric key encryption SKE. As in the case of DFA, the function key selects the appropriate SKE keys to reveal the CktFE ciphertext encoding next state and symbol to be read.

The careful reader may have noticed that the above description glosses over an important detail: the cell that is written into at step i may be next accessed at any step $j > i$, so the CktFE ciphertext at step i must encode SKE keys for some unknown future step j. Fortunately, the machinery of oblivious TMs comes to our aid again. Since in an oblivious TM, there exists a function t that computes the step that particular cell will be accessed next, in step i, in addition to selecting the state for step $i+1$ as we did in DFAs, the function key will also select the tape symbol to be read in step $t(i)$. At any step j, the appropriate SKE keys for the state were provided in step $j-1$ and for tape symbol were provided at step $i < j$ where $t(i) = j$. Hence, the decryptor at step j has the SKE keys to unlock the CktFE CT components for both state and tape symbol, which lets her proceed with the computation. For more details, please see Appendix D.

1.4 Organization of the paper.

In Appendix A, we discuss additional related work. In Section 2, we define the preliminaries we require for our constructions. In Section 3, we define the notion of decomposable functional encryption. In Section 4, we provide our construction for single key FE for DFAs. We show how to compile this with symmetric key encryption to achieve reusable garbled DFAs in Appendix E. In Appendix D, we provide our construction for single key functional encryption for Turing machines.

2 Definitions : FE for Deterministic Finite Automata

In this section we provide some notation and preliminaries that we require. In Appendix B, we provide notation and additional definitions.

Functional encryption for deterministic finite automata (DFA) is defined analogously to functional encryption for circuits, except that the public parameters may not depend on the input length, which is unknown a priori. In this section, we will define single key functional encryption for DFAs.

A DFA machine M is represented by the tuple $(Q, \Sigma, T, q_{st}, F)$ where Q is a finite set of states, Σ is a finite alphabet, $T : \Sigma \times Q \to Q$ is the transition function (stored as a table), q_{st} is the start state, $F \subseteq Q$ is the set of accepting states. Upon input $\mathbf{w} \in \Sigma^k$ for some arbitrary polynomial k (not known to the setup algorithm), the machine M accepts the input if and only if there exists a sequence of states $q_1, \ldots, q_k$ so that $q_1 = q_{st}$, $T(w_i, q_i) = q_{i+1}$ for $i \in [k-1]$, and $q_k \in F$. We say $M(\mathbf{w}) = 1$ iff M accepts $\mathbf{w}$ and 0 otherwise.

2.1 Definition

Let $\mathcal{M}_\kappa : \mathcal{Q}_\kappa \times \Sigma_\kappa \to \mathcal{Q}_\kappa$ be a DFA family. A functional encryption scheme DfaFE for $\mathcal{M}$ consists of four algorithms DfaFE = (DfaFE.Setup, DfaFE.KeyGen, DfaFE.Enc, DfaFE.Dec) defined as follows.

- $\mathsf{DfaFE.Setup}(1^\kappa)$ is a p.p.t. algorithm takes as input the unary representation of the security parameter and outputs the master public and secret keys $(\mathsf{PK}, \mathsf{MSK})$.
- $\mathsf{DfaFE.KeyGen}(\mathsf{MSK}, \mathsf{M})$ is a p.p.t. algorithm that takes as input the master secret key MSK and a DFA machine M and outputs a corresponding secret key SK_M.
- $\mathsf{DfaFE.Enc}(\mathsf{PK}, \mathbf{w})$ is a p.p.t. algorithm that takes as input the master public key PK and an input message $\mathbf{w}$ and outputs a ciphertext $\mathsf{CT}_\mathbf{w}$.
- $\mathsf{DfaFE.Dec}(\mathsf{SK}_\mathsf{M}, \mathsf{CT}_\mathbf{w})$ is a deterministic algorithm that takes as input the secret key SK_M and a ciphertext $\mathsf{CT}_\mathbf{w}$ and outputs $M(\mathbf{w})$.

▶ **Definition 1** (Correctness)**.** A functional encryption scheme DfaFE is correct if for all $M \in \mathcal{M}$ and all $\mathbf{w} \in \Sigma^*$,

$$\Pr\left[\begin{array}{l}(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{DfaFE.Setup}(1^\kappa);\\ \mathsf{DfaFE.Dec}\Big(\mathsf{DfaFE.KeyGen}(\mathsf{MSK},\mathsf{M}), \mathsf{DfaFE.Enc}(\mathsf{PK},\mathbf{w})\Big) \neq \mathsf{M}(\mathbf{w})\end{array}\right] = \mathrm{negl}(\kappa)$$

where the probability is taken over the coins of DfaFE.Setup, DfaFE.KeyGen, and DfaFE.Enc.

2.2 Security

In this section, we define simulation based security for single key FE for DFAs. The definition is analogous to that for circuits, as provided in Appendix B.1.1.

▶ **Definition 2** (FULL-SIM- Security for DFA-FE.)**.** Let $\mathcal{F}_\mathcal{M}$ be a functional encryption scheme for a DFA family $\mathcal{M}$. For every p.p.t. adversary $A = (A_1, A_2)$ and a p.p.t. simulator Sim, consider the following two experiments:

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{DfaFE},\mathsf{A}}(1^\kappa)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{DfaFE},\mathrm{Sim}}(1^\kappa)$:
1: $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{DfaFE.Setup}(1^\kappa)$	1: $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{DfaFE.Setup}(1^\kappa)$
2: $(M, st_1) \leftarrow A_1(\mathsf{PK})$	2: $(M, st_1) \leftarrow A_1(\mathsf{PK})$
3: $\mathsf{sk}_M \leftarrow \mathsf{DfaFE.KeyGen}(\mathsf{MSK},\mathsf{M})$	3: $\mathsf{sk}_M \leftarrow \mathsf{DfaFE.KeyGen}(\mathsf{MSK},\mathsf{M})$
4: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_M)$	4: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_M)$
5: $\mathsf{CT} \leftarrow \mathsf{DfaFE.Enc}(\mathsf{PK}, \mathbf{x})$	5: $\tilde{\mathsf{CT}} \leftarrow \mathrm{Sim}(\mathsf{PK}, \mathsf{sk}_M, M, M(\mathbf{x}), 1^{\lvert\mathbf{x}\rvert})$
6: Output (st, CT)	6: Output $(st, \tilde{\mathsf{CT}})$

The DFA functional encryption scheme $\mathcal{F}_\mathcal{M}$ is then said to be single query FULL-SIM secure if there exists a p.p.t. simulator Sim such that for every p.p.t. adversary $A = (A_1, A_2)$, the following two distributions are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{DfaFE},\mathsf{A}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{DfaFE},\mathrm{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

3 Decomposable Functional Encryption for Circuits

In this section, we define the notion of *decomposable functional encryption* (DFE). Decomposable functional encryption is analogous to the notion of decomposable randomized encodings [3]. 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.

We assume that given the security parameter, the following spaces are fixed: $\mathcal{P}$ containing public key components, $\mathcal{R}_1$, $\mathcal{R}_2$ containing randomness used for encryption and $\mathcal{C}$ containing the encoding of a single message bit. The length of the message $|\mathbf{x}|$ can be any polynomial. Formally, let $\mathbf{x} \in \{0,1\}^k$. A functional encryption scheme is said to be decomposable if there exists a deterministic function $\mathcal{E} : \mathcal{P} \times \{0,1\} \times \mathcal{R}_1 \times \mathcal{R}_2 \to \mathcal{C}$ 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 $\hat{r}_i \in \mathcal{R}_2$.

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. $\bar{\mathcal{E}}(\mathsf{PK}, \mathbf{y}, r) \triangleq \left(\mathcal{E}(\mathsf{PK}_1, y_1, r, \hat{r}_1), \ldots, \mathcal{E}(\mathsf{PK}_k, y_k, r, \hat{r}_k)\right)$, where $|\mathbf{y}| = k$.

4 Single-Key Succinct FE for DFAs from LWE

In this section, we will construct a single key (public key) functional encryption scheme for deterministic finite automata (DFA). Our construction makes use a decomposable single key FE scheme for circuits, CktFE. In Appendix C.1, we show that:

▶ **Lemma 3.** *The single key, succinct functional encryption scheme for circuits by Goldwasser et al. [18], based on LWE (Appendix B.4) is decomposable.*

Conceptually, we decompose the input into two components of size ℓ_1 and ℓ_2 each, where the second component is further decomposed bit by bit. We will use the first component to encrypt the current input symbol in the DFA computation and keys to select the next state in the computation, and the second component to encrypt the current state in the DFA computation. While the input symbol encoded in the first component can be treated as a unit of size ℓ_1, it will be helpful for us to represent the encoded input of size ℓ_2 bit by bit.

Thus, we have,

$$\mathsf{CktFE.PK} = (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{PK}_{\mathsf{indpt}}) \text{ and } \mathsf{CktFE.CT} = (\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}_{\mathsf{indpt}})$$

$$\begin{aligned} \text{Now, let } \mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x}\|\mathbf{y}) &= (\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}_{\mathsf{indpt}}) \\ &= \left(\bar{\mathcal{E}}(\mathsf{PK}_1, \mathbf{x}, r, \hat{\mathbf{r}}_1),\ \bar{\mathcal{E}}(\mathsf{PK}_2, \mathbf{y}, r, \hat{\mathbf{r}}_2),\ \bar{\mathcal{E}}(\mathsf{PK}_{\mathsf{indpt}}, r, \hat{\mathbf{r}}_3)\right) \\ \text{We decompose } \bar{\mathcal{E}}(\mathsf{PK}_2, \mathbf{y}, r, \hat{\mathbf{r}}_2) &= \left(\mathcal{E}(\mathsf{PK}_{2,1}, y_1, r, \hat{r}_{2,1}), \ldots, \mathcal{E}(\mathsf{PK}_{2,\ell_2}, y_{\ell_2}, r, \hat{r}_{2,\ell_2})\right) \end{aligned}$$

Recall that $\mathcal{E} : \mathcal{P} \times \{0,1\} \times \mathcal{R}_1 \times \mathcal{R}_2 \to \mathcal{C}$ and $\bar{\mathcal{E}}(\mathbf{x})$ denotes the tuple of function values obtained by applying $\mathcal{E}$ to each coordinate. Then,

$$\text{Let} \quad |\mathbf{x}| = \ell_1, \quad |\mathbf{y}| = \ell_2, \quad \mathsf{PK}_1 \in \mathcal{P}^{\ell_1}, \quad \mathsf{PK}_2 \in \mathcal{P}^{\ell_2},$$
$$j \in \mathbb{N}, \quad \mathsf{PK}_{\mathsf{indpt}} \in \mathcal{P}^j, \quad r \in \mathcal{R}_1, \quad \hat{\mathbf{r}}_1 \in \mathcal{R}_2^{\ell_1}, \quad \hat{\mathbf{r}}_2 \in \mathcal{R}_2^{\ell_2}, \quad \hat{\mathbf{r}}_3 \in \mathcal{R}_2^j$$

In what follows, we abuse notation slightly and omit mention of the independent, fresh randomness from $\mathcal{R}_2$ needed for each invocation for $\mathcal{E}$. For convenience, we club the message independent component $\mathsf{CT}_{\mathsf{indpt}}$ with CT_1 and let

$$\mathbf{c} = (\mathsf{CT}_1, \mathsf{CT}_{\mathsf{indpt}}) \text{ and } \mathbf{d} = \mathsf{CT}_2 = (\mathsf{CT}_{2,1}, \ldots, \mathsf{CT}_{2,\ell_2})$$

Let $\mathcal{M}_\kappa : \mathcal{Q}_\kappa \times \Sigma_\kappa \to \mathcal{Q}_\kappa$ be a DFA family. For notational convenience, we will drop the subscript κ here on. Let $Q = |\mathcal{Q}|$, the size of the state space of the DFA family. Then, the single key functional encryption scheme for DFAs is constructed as follows.

$\mathsf{DfaFE.Setup}(1^\kappa)$: Upon input the security parameter 1^κ, do:

1. Choose a symmetric key encryption scheme SKE with key space $\mathcal{K}$.
2. Define a circuit family as follows. Let $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} = (\Sigma \times \mathcal{K}^{2\log Q} \times \{0,1\}) \times \mathcal{Q}$ and $\mathcal{Y} = \mathcal{K}^{\log Q}$. We set
$$\ell_1 = \lfloor \Sigma \rfloor + \lfloor \mathcal{K}^{2\log Q} \rfloor + 1, \quad \ell_2 = \lfloor \mathcal{Q} \rfloor = \log Q$$
where $\lfloor \cdot \rfloor$ denotes size in bits. Let $\ell = \ell_1 + \ell_2$.
3. Invoke $\mathsf{CktFE.Setup}(1^\kappa, 1^\ell)$ to obtain $\mathsf{PK} = \big(\mathsf{PK}_1, (\mathsf{PK}_{2,1}, \ldots, \mathsf{PK}_{2,\log Q}), \mathsf{PK}_{\mathsf{indpt}}\big)$ and MSK.
4. Output $(\mathsf{PK}, \mathsf{MSK})$.

$\mathsf{DfaFE.Enc}(\mathsf{PK}, \mathbf{w})$: Let $|\mathbf{w}| = k$. Note that k is arbitrary, and unknown to $\mathsf{DfaFE.Setup}$. Do the following:

1. Sample randomness $r_i \leftarrow \mathcal{R}_1$ for $i \in [k]$.
2. Sample SKE keys as follows. We define
$$\mathbf{K}_{i+1} = \Big((K^0_{(i+1,1)}, K^1_{(i+1,1)}), \ldots, (K^0_{(i+1,\log Q)}, K^1_{(i+1,\log Q)})\Big)$$
where $K^b_{i+1,j} \leftarrow \mathcal{K}$ for $i \in [k-1]$, $j \in [\log Q]$, $b \in \{0,1\}$.
3. Define message $\mathbf{y}_i = (w_i, \mathbf{K}_{i+1}, 0)$ for $i \in [k-1]$ and $\mathbf{y}_k = (w_k, \perp, 1)$.
4. For $i \in [k]$, we define:
$$\mathbf{c}_{i,1} = \bar{\mathcal{E}}(\mathsf{PK}_1, \mathbf{y}_i, r_i), \quad \mathbf{c}_{i,2} = \bar{\mathcal{E}}(\mathsf{PK}_{\mathsf{indpt}}, r_i), \quad \mathbf{c}_i = (\mathbf{c}_{i,1}, \mathbf{c}_{i,2})$$
5. Let $\mathbf{d}_1 = \bar{\mathcal{E}}\ (\mathsf{PK}_2, q_{st}, r_1)$. Here q_{st} denotes the start state of the DFA. Further, let
$$\mathbf{d}^b_{i,j} = \mathcal{E}\ (\mathsf{PK}_{2,j}, b, r_i)\ \ \forall\ i \in [2,k],\ j\ \in [\log Q],\ b \in \{0,1\}.$$
$$\mathbf{d}_{i,q} \triangleq (\mathbf{d}^{q_j}_{i,j})\ \ \forall j \in [\log Q] \text{ where } q_j \text{ is the } j\text{th bit of } q.$$
6. For $i \in [2,k]$, $j \in [\log Q]$, $b \in \{0,1\}$ encrypt each $\mathbf{d}^b_{i,j}$ using the corresponding key $K^b_{i,j}$ as:
$$\hat{\mathbf{d}}^b_{i,j} = \mathsf{SKE.Enc}(K^b_{i,j}, \mathbf{d}^b_{i,j})$$
7. Choose $b_{i,j} \leftarrow \{0,1\}$ randomly for $i \in [2,k]$, $j \in [\log Q]$ and define:
$$\hat{\mathbf{D}}_{i,j} = (\ \hat{\mathbf{d}}^{b_{i,j}}_{i,j}, \hat{\mathbf{d}}^{\bar{b}_{i,j}}_{i,j}\ ), \quad \hat{\mathbf{D}}_i = (\hat{\mathbf{D}}_{i,j}), \quad \hat{\mathbf{D}}_1 = \mathbf{d}_1$$
8. Output $\mathsf{CT}_{\mathbf{w}} = \{\mathbf{c}_i, \hat{\mathbf{D}}_i\}$ for $i \in [k]$.

$\mathsf{DfaFE.KeyGen}(\mathsf{MSK}, \mathsf{M})$: Let M denote a DFA machine and T denote its transition matrix. Let T_i denote the i^{th} row of T, with format $\big((\sigma, q) \to q'\big)$ indicating that the machine enters state q' upon input symbol σ and input state q. Let $\mathsf{SK}_M = \mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$ where f is defined below in Figure 1.

$\mathsf{DfaFE.Dec}(\mathsf{SK}_f, \mathsf{CT}_{\mathbf{w}})$: Interpret $\mathsf{CT}_{\mathbf{w}} = (\mathbf{c}_i, \hat{\mathbf{D}}_i)_{i\in[k]}$ and let $\mathbf{d}_{1,q_1} = \hat{\mathbf{D}}_1$.
Initialize $i = 1$. While $i \le k$, do the following:

- Let $\mathsf{CT}'_i = (\mathbf{c}_i, \mathbf{d}_{i,q_i})$. Recall that $\mathbf{d}_{i,q_i} = (\mathbf{d}^{q_{i,j}}_{i,j})$ for $j \in [\log Q]$. If $i = k$, let $b \leftarrow \mathsf{CktFE.Dec}(\mathsf{SK}_f, \mathsf{CT}'_k)$. Output b and exit.
- Else let $(K_{i+1,1}, \ldots, K_{i+1,\log Q}) = \mathsf{CktFE.Dec}(\mathsf{SK}_f, \mathsf{CT}'_i)$.

Function $f\big((\sigma, \mathbf{K}, \mathsf{flag}), q\big)$

1. Lookup table T for (σ, q). Say that $(\sigma, q) \rightarrow q'$. If no entry is found, output $\perp$ and exit.
2. If $\mathsf{flag} = 1$, check if q' is an accepting state. If yes, output 1, else output 0 and exit.
3. If $\mathsf{flag} = 0$, parse $\mathbf{K}$ as $\{(K_j^0, K_j^1)\}$ for $j \in [\log Q]$, $b \in \{0,1\}$. Choose the $\log Q$ keys $K_j^{q'_j}$ (for $j \in [\log Q]$), corresponding to the bits of q' and output these.

Figure 1 Function to provide keys for next state in DFA computation.

- For $j \in [\log Q]$, try to decrypt each value in $\hat{\mathbf{D}}_{i+1,j}$ using obtained key $K_{i+1,j}$. Exactly one of the two ciphertexts per bit position will be decrypted, say $\hat{\mathbf{d}}_{i+1,j}^{b_j}$. Set

$$\mathbf{d}_{i+1,q_{i+1}} = \Big(\mathsf{SKE.Dec}(K_{i+1,j}, \hat{\mathbf{d}}_{i+1,j}^{b_j}) \Big) \quad \forall j \in [\log Q]$$

- Increment i.

4.1 Correctness.

In this section, we establish correctness of the above construction. Before we proceed with the formal argument, we provide some intuition. Note that in the encryption, the first component $\mathbf{c}_i$ encrypts message $\mathbf{y}_i$, which contains the ith input symbol, along with the set of all $2 \log Q$ symmetric keys used to construct SKE encryptions of the $(i+1)^{th}$ state. In the second component, the element $\mathbf{d}_{i,j}^b$ in tuple $(\ \mathbf{d}_{i,j}^b\ )$ for $j \in [\log Q]$ and $b \in \{0,1\}$, contains an encryption of bit b, corresponding to the event that the j^{th} bit of i^{th} state is b. The set $\hat{\mathbf{D}}_i$ contains $2 \log Q$ SKE encryptions of $\mathbf{d}_{i,j}^b$ under keys $K_{i,j}^b$, shuffled for each position j. Decryption at step $i-1$ provides the level i symmetric keys $K_{i,j}^{b_j}$ to unlock the $\mathbf{d}_{i,j}^{b_j}$ for the correct next state of the computation q', i.e. $b_j = q'_{i,j}$. Thus, the decryptor recovers exactly the components $\mathbf{d}_{i,j}^{b_j}$ which may be combined to create the ciphertext $\mathbf{d}_{i,q_i}$. Put together with $\mathbf{c}_i$ we get an encryption of $(w_i, \mathbf{K}_{i+1}, q_i)$ which may again be decrypted with the function key to obtain the appropriate keys to decrypt the correct $\mathbf{d}_{i+1,q_{i+1}}$.

Formally, let k denote the length of input $\mathbf{w}$ and let $q_1, \ldots, q_k$ denote the states visited by the DFA during computation. We have by correctness of decomposable functional encryption that:

$$\forall i \in [k-1],\ \mathsf{CktFE.Enc}\big(\ \mathsf{PK}, (w_i, \mathbf{K}_{i+1}, 0, q_i)\ \big) = (\mathbf{c}_i, \mathbf{d}_{i,q_i}) \quad \text{where}$$

$$\mathbf{c}_k = \Big(\bar{\mathcal{E}}\big(\mathsf{PK}_1, (w_i, \mathbf{K}_{i+1}, 0), r_i, \big), \bar{\mathcal{E}}\big(\mathsf{PK}_{\mathsf{indpt}}, r_i\big)\Big), \quad \mathbf{d}_{i,q_i} = \big(\mathcal{E}(\mathsf{PK}_{2,j}, q_{i,j}, r_i)\big)_{j \in [\log Q]}$$

$$\text{s.t. } \mathsf{CktFE.Dec}\big(\mathsf{SK}_f, (\mathbf{c}_i, \mathbf{d}_{i,q_i})\big) = \mathbf{K}_{q_{i+1}} \triangleq (K_{i+1,1}^{b_1}, \ldots, K_{i+1,\log Q}^{b_{\log Q}}) \ \text{ where } \ b_j = q_{i+1,j}.$$

Now, both elements of $\hat{\mathbf{D}}_{i+1,j}$ are attempted for decryption by $K_{i+1,j}^{b_j}$, of which only the element encoding the correct bit $q_{i+1,j}$ is recovered. Formally, we have:

$$\hat{\mathbf{D}}_{i+1,j} = (\ \hat{\mathbf{d}}_{i+1,j}^0, \hat{\mathbf{d}}_{i+1,j}^1\ ) \quad \text{and}$$

$$\mathsf{SKE.Dec}\ (\ K_{i+1,j}^{b_j},\ \hat{\mathbf{d}}_{i+1,j}^b\ ) = \perp \ \text{ if } \ b_j \neq b, \text{ and } \mathbf{d}_{i+1,j}^{q_{i+1,j}} \quad \text{otherwise.}$$

By putting together all the components, we get by decomposability:

$$\mathbf{d}_{i+1,q_{i+1}} = (\ \mathbf{d}_{i+1,j}^{q_{i+1,j}}\ ) \ \ \forall\ j \in [\log Q]$$

Also, since each component of $\mathbf{d}_{i+1,q_{i+1}}$ uses the same common randomness r_{i+1} as is used by $\mathbf{c}_{i+1}$, we have that $\mathsf{CT}_{i+1} = (\mathbf{c}_{i+1}, \mathbf{d}_{i+1,q_{i+1}})$, hence we may repeat while $i < k$. Finally for $i = k$,

$$\mathsf{CktFE.Enc}\big(\ \mathsf{PK}, (w_k, \bot, 1, q_k)\ \big) = (\mathbf{c}_i, \mathbf{d}_{k,q_k})$$

so that $\mathsf{CktFE.Dec}\big(\mathsf{SK}_f, (\mathbf{c}_k, \mathbf{d}_{k,q_k})\big) = 1$ iff q_k is an accepting state, 0 otherwise.

Efficiency. We note that the public key size of our scheme is the public key size of CktFE [18] with message length $\ell = O(\log Q + \log|\Sigma| + \log Q \cdot \log|\mathcal{K}|$ which is polynomial in the security parameter κ. The ciphertext size is $O(|\mathbf{w}| \cdot \log Q)$ and the secret key size is $O(|M|)$ (ignoring polynomials in the security parameter).

4.2 Proof of Security

We proceed to show that our construction is secure. Formally,

▶ **Theorem 4.** *Assume that the underlying* CktFE *scheme satisfies* FULL-SIM *security according to definition 7. Then the construction for* DfaFE *achieves* FULL-SIM *security as defined in definition 2.*

Proof. We proceed to construct a simulator DfaFE.Sim as required by Definition 2. The simulator receives $(\mathsf{PK}, \mathsf{SK}_M, M, M(\mathbf{w}), 1^{|\mathbf{w}|})$ and does the following:

1. Assign the bit $b = M(\mathbf{w})$, and construct the circuit f corresponding to M as defined in Figure 1 in the description of DfaFE.KeyGen.
2. Let $\mathsf{CktFE.SK_f} = \mathsf{SK_M}$ and invoke $\mathsf{CktFE.Sim}\big(\ \mathsf{PK}, f, \mathsf{CktFE.SK}_f, b\ \big)$ to receive $\tilde{\mathsf{CT}}_k$ where we may express $\tilde{\mathsf{CT}}_k = (\tilde{\mathbf{c}}_k, \tilde{\mathbf{d}}_{k,q_k})$ and $\tilde{\mathbf{d}}_{k,q_k} = (\tilde{\mathbf{d}}_{k,j})$ for $j \in [\log Q]$.
3. For $(i = k, i \geq 1, i--)$, do:
   a. If $i = 1$, set $\mathsf{Sim}.\hat{\mathbf{D}}_1 = \tilde{\mathbf{d}}_{1,q_1}$ and exit.
   b. Sample key $\mathbf{K}^*_i = (K^*_{i,1}, \ldots, K^*_{i,\log Q}) \leftarrow \mathcal{K}^{\log Q}$ and let
   $$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j} = \mathsf{SKE.Enc}(K^*_{i,j}, \tilde{\mathbf{d}}_{i,j})\ \ \forall j \in [\log Q],$$
   c. Sample $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and assign $\mathsf{Sim}.\hat{\mathbf{d}}^{\tilde{b}_{i,j}}_{i,j} = \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}$.
   d. Choose another $\log Q$ keys $\tilde{K}_{i,1}, \ldots \tilde{K}_{i,\log Q} \leftarrow \mathcal{K}^{\log Q}$ and compute
   $$\mathsf{Sim}.\hat{\mathbf{d}}^{\bar{b}_{ij}}_{i,j} = \mathsf{SKE.Enc}(\tilde{K}_{i,j}, 0^{|\tilde{\mathbf{d}}_{i,j}|})\ \ \forall\ j \in [\log Q]$$
   e. Let $\mathsf{Sim}.\hat{\mathbf{D}}_{i,j} = \big(\mathsf{Sim}.\hat{\mathbf{d}}^{\tilde{b}_{i,j}}_{i,j},\ \mathsf{Sim}.\hat{\mathbf{d}}^{\bar{b}_{i,j}}_{i,j}\big)$ and $\mathsf{Sim}.\hat{\mathbf{D}}_i = \big(\ \mathsf{Sim}.\hat{\mathbf{D}}_{i,j}\ \big)$ for $j \in [\log Q]$.
   f. Let $(\tilde{\mathbf{c}}_{i-1}, \tilde{\mathbf{d}}_{i-1,q_{i-1}}) = \mathsf{CktFE.Sim}(\mathsf{PK}, f, \mathsf{SK}_f, \mathbf{K}^*_i)$.
4. Output the ciphertext as $\mathsf{CT_w} = (\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2, \ldots, \tilde{\mathbf{c}}_k, \mathsf{Sim}.\hat{\mathbf{D}}_1, \ldots, \mathsf{Sim}.\hat{\mathbf{D}}_k)$.

4.2.1 Analysis of Simulator.

Correctness of the simulator DfaFE.Sim can be easily established using correctness of the simulator CktFE.Sim and the semantic security of SKE. Let us say that the DFA M visits states $q_1, \ldots, q_k$ while computing on input $\mathbf{w}$ where $|\mathbf{w}| = k$.

1. We have by correctness of CktFE.Sim according to definition 7 that:

$$\left\{\mathsf{CT}_k \leftarrow \mathsf{CktFE.Enc}\big(\mathsf{PK}, (w_k, \perp, 1, q_k)\big) \stackrel{c}{\approx} \tilde{\mathsf{CT}}_k \leftarrow \mathsf{CktFE.Sim}\big(\mathsf{PK}, f_M, \mathsf{SK}_f, b\big)\right\}$$

   By decomposability, $\mathsf{CT}_k = (\mathbf{c}_k, \mathbf{d}_{k,q_k})$ where $\mathbf{d}_{k,q_k} = (\mathbf{d}_{k,j}^{b_j})$ for $j \in [\log Q]$ and $b_j = q_{k,j}$ defined as the j^{th} bit of state q_k. Similarly, $\tilde{\mathsf{CT}}_k = (\tilde{\mathbf{c}}_k, \tilde{\mathbf{d}}_{k,q_k})$ where $\tilde{\mathbf{d}}_{k,q_k}$ may be decomposed as $(\tilde{\mathbf{d}}_{k,j})$ for $j \in [\log Q]$. Let $i = k$.
2. We now establish that $(\hat{\mathbf{d}}_{i,j}^{b_j} \stackrel{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}_{i,j})$ where $b_j = q_{i,j}$ and $j \in [\log Q]$.
   a. We have that in algorithm DfaFE.Enc,

$$\mathbf{K}_i = \Big((K^0_{(i,1)}, K^1_{(i,1)}), \ldots, (K^0_{(i,\log Q)}, K^1_{(i,\log Q)})\Big)$$

   where $K^b_{i,j} \leftarrow \mathcal{K}$ for $j \in [\log Q]$. We also have, for $j \in [\log Q]$, $b \in \{0,1\}$:

$$\hat{\mathbf{d}}^b_{i,j} = \mathsf{SKE.Enc}(K^b_{i,j}, \mathbf{d}^b_{i,j}) \tag{4.1}$$

   b. In simulator DfaFE.Sim:

$$\mathbf{K}^*_i = (K^*_{i,1}, \ldots, K^*_{i,\log Q}) \leftarrow \mathcal{K}^{\log Q} \quad \text{and}$$

$$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j} = \mathsf{SKE.Enc}(K^*_{i,j}, \tilde{\mathbf{d}}_{i,j}) \;\; \forall j \in [\log Q],$$

   Hence, since $\mathbf{d}^{b_j}_{i,j} \stackrel{c}{\approx} \tilde{\mathbf{d}}_{i,j}$ and the symmetric keys are picked using the same distribution in each case, we have that $(\hat{\mathbf{d}}^{b_j}_{i,j} \stackrel{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}_{i,j})$ where $b_j = q_{i,j}$ and $j \in [\log Q]$.
3. We now establish that $(\hat{\mathbf{d}}^{\bar{b}_j}_{i,j} \stackrel{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}^{\bar{b}_j}_{i,j})$ where $j \in [\log Q]$.
   a. Construction of $\hat{\mathbf{d}}^{\bar{b}_j}_{i,j}$ is described in Equation 4.1.
   b. For the latter, DfaFE.Sim samples $\tilde{b}_j$ and sets $\mathsf{Sim}.\hat{\mathbf{d}}^{\tilde{b}_j}_{i,j} = \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}$. Next, it samples another $\log Q$ keys $\tilde{K}_{i,1}, \ldots \tilde{K}_{i,\log Q} \leftarrow \mathcal{K}^{\log Q}$ and computes

$$\mathsf{Sim}.\hat{\mathbf{d}}^{\bar{\tilde{b}}_j}_{i,j} = \mathsf{SKE.Enc}(\tilde{K}_{i,j}, 0^{|\tilde{\mathbf{d}}_{i,j}|}) \;\; \forall\, j \in [\log Q]$$

   By semantic security of SKE, we have that $(\hat{\mathbf{d}}^{\bar{b}_j}_{i,j} \stackrel{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}^{\bar{\tilde{b}}_j}_{i,j})$.
4. Next, we show that $\hat{\mathbf{D}}_i \stackrel{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$. For $i = 1$, we have by definitions of $\hat{\mathbf{D}}_1$ and $\mathsf{Sim}.\hat{\mathbf{D}}_1$, that the above holds. For $i > 1$, in DfaFE.Enc, we have $b_{i,j} \leftarrow \{0,1\}$ and

$$\hat{\mathbf{D}}_{i,j} = \big(\, \hat{\mathbf{d}}^{b_{i,j}}_{i,j}, \hat{\mathbf{d}}^{\bar{b}_{i,j}}_{i,j} \,\big)$$

   In DfaFE.Sim, we have $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and

$$\mathsf{Sim}.\hat{\mathbf{D}}_{i,j} = \big(\mathsf{Sim}.\hat{\mathbf{d}}^{\tilde{b}_{i,j}}_{i,j},\; \mathsf{Sim}.\hat{\mathbf{d}}^{\bar{\tilde{b}}_{i,j}}_{i,j}\big)$$

   Since $\hat{\mathbf{D}}_i = (\hat{\mathbf{D}}_{i,j})$ and $\mathsf{Sim}.\hat{\mathbf{D}}_i = \big(\, \mathsf{Sim}.\hat{\mathbf{D}}_{i,j} \,\big)$ for $j \in [\log Q]$, we have that $\hat{\mathbf{D}}_i \stackrel{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$.
5. Let $i = i - 1$. Now, we have by correctness of CktFE.Sim according to Definition 7,

$$\left\{\mathsf{CT}_i \leftarrow \mathsf{CktFE.Enc}\big(\mathsf{PK}, (w_i, \mathbf{K}_{i+1}, 0, q_i)\big) \stackrel{c}{\approx} \tilde{\mathsf{CT}}_i \leftarrow \mathsf{CktFE.Sim}\big(\mathsf{PK}, f_M, \mathsf{SK}_f, \mathbf{K}^*_{i+1}\big)\right\}$$

   By decomposability, $\mathsf{CT}_i = (\mathbf{c}_i, \mathbf{d}_{i,q_i})$ where $\mathbf{d}_{i,q_i} = (\mathbf{d}^{q_{i,j}}_{i,j})$ for $j \in [\log Q]$. Also, $\tilde{\mathsf{CT}}_i = (\tilde{\mathbf{c}}_i, \tilde{\mathbf{d}}_{i,q_i})$ where $\tilde{\mathbf{d}}_{i,q_i} = (\tilde{\mathbf{d}}_{i,j})$ for $j \in [\log Q]$. If $i > 1$, go to step 2. For $i = 1$, we have by definitions of $\hat{\mathbf{D}}_1$ and $\mathsf{Sim}.\hat{\mathbf{D}}_1$, that $(\mathbf{c}_1, \hat{\mathbf{D}}_1) \stackrel{c}{\approx} (\tilde{\mathbf{c}}_1, \mathsf{Sim}.\hat{\mathbf{D}}_1)$.

6. Now, a straightforward hybrid argument yield that:

$$\left\{(\mathbf{c}_1, \hat{\mathbf{D}}_1), (\mathbf{c}_2, \hat{\mathbf{D}}_2), \ldots, (\mathbf{c}_k, \hat{\mathbf{D}}_k)\right\} \overset{c}{\approx} \left\{(\tilde{\mathbf{c}}_1, \mathsf{Sim}.\hat{\mathbf{D}}_1), (\tilde{\mathbf{c}}_2, \mathsf{Sim}.\hat{\mathbf{D}}_2), \ldots, (\tilde{\mathbf{c}}_k, \mathsf{Sim}.\hat{\mathbf{D}}_k)\right\}$$

as desired.

◀

Reusable Garbled DFA. In Appendix E we show how to compile the above construction with symmetric key encryption to obtain the first construction of reusable garbled DFAs from standard assumptions.

5 Single Key Functional Encryption for Turing Machines

In Appendix D, we provide the construction of single key functional encryption for Turing machines. Our construction has short public parameters that are independent of the size of the machine or the data being encrypted, short function keys, and input-specific decryption time. However, the ciphertext of our construction is large and depends on the worst case running time of the Turing machine (but not its description size).

While the large ciphertext size of our TMFE construction restricts its utility for practical applications, we emphasize that the parameters obtained by our TMFE construction are not implied by previous work to the best of our knowledge (please see Appendix A for a detailed discussion about previous work). To improve the ciphertext size of our construction, while allowing succinct keys, dynamic data length and input specific run time is an interesting open problem.

REFERENCES AND SUPPLEMENTARY MATERIAL

References

1 Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, pages 308–326. Springer, 2015.

2 Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography*, pages 125–153. Springer, 2016.

3 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.

4 Sanjeev Arora and Boaz Barak. Complexity theory: A modern approach. *Online draft at http://www. cs. princeton. edu/theory/complexity*, 2008.

5 Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, 2014.

6 Domagoj Babić, Daniel Reynaud, and Dawn Song. Malware analysis with tree automata inference. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, 2011.

7 Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

8 Xavier Boyen and Qinyi Li. Attribute-based encryption for finite automata from lwe. In *Provable Security*, pages 247–267. Springer, 2015.

9 Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

10 Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13. ACM, 2013.

11 Zvika Brakerski and Vinod Vaikuntanathan. Circuit-abe from lwe: Unbounded attributes and semi-adaptive security. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, 2016.

12 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.

13 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.

14 Sanjeev Das, Hao Xiao, Yang Liu, and Wei Zhang. Online malware defense using attack behavior model. In *IEEE International Symposium on Circuits and Systems, ISCAS 2016, Montréal, QC, Canada, May 22-25, 2016*, pages 1322–1325, 2016.

15 Jean H Gallier. *Logic for computer science: foundations of automatic theorem proving*. Courier Dover Publications, 2015.

16 Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 449–458. ACM, 2015.

17 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.

18 Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

19 Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute based encryption for circuits. In *STOC*, 2013.

20 Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. Cryptology ePrint Archive, Report 2015/029, 2015. `http://eprint.iacr.org/`.

21 Christopher L. Hayes and Yan Luo. Dpico: A high speed deep packet inspection engine using compact finite automata. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, ANCS '07, pages 195–203, 2007.

22 Russell Impagliazzo. Notes on turing machines. `http://cseweb.ucsd.edu/classes/sp11/cse201A-a/ln412.pdf`.

23 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.

24 Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, 2012.

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

26 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J.ACM*, 56(6), 2009. extended abstract in STOC'05.

27 Amit Sahai and Hakan Seyalioglu. Worry-free encryption: Functional encryption with public keys. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010.

28 Daniele Paolo Scarpazza, Oreste Villa, and Fabrizio Petrini. Peak-performance dfa-based string matching on the cell processor. In *21th International Parallel and Distributed Processing Symposium (IPDPS), Proceedings, 26-30 March 2007, Long Beach, California, USA*, pages 1–8, 2007.

29 Brent Waters. Functional encryption for regular languages. In *Crypto*, 2012.

30 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

APPENDICES

A Related work

The following alternate method can be used to obtain a single key functional encryption scheme for Turing machines based on the hardness of public key encryption: replace the garbled circuit in the construction of [27] by a garbled TM (see [16] and follow-ups). In more detail, let the length of Turing machines in the given family be bounded by some polynomial ℓ_M. The public key for the TMFE scheme consists of $2 \cdot \ell_M$ public keys $\mathsf{PK}_{i,b}$ for $i \in [\ell_M]$, $b \in \{0,1\}$. The encryptor, given $\mathbf{w}$ of arbitrary length constructs a garbling of the universal TM, and additionally provides garbled labels for $\mathbf{w}$ in the clear and public key encryptions of the labels of all possible bits of the TM M. Specifically, the encryptor provides public key encryptions of labels of both bits $b \in \{0,1\}$ corresponding to each position $i \in [\ell_M]$ under public key $\mathsf{PK}_{i,b}$. The master secret key for a machine M is the secret keys SK_{i,M_i} where

M_i is the i^{th} bit of M. The decryptor uses the secret keys to recover the garbled labels for M and runs the garbled TM on the garbled labels corresponding to $\mathbf{w}$ and M to recover $M(\mathbf{w})$. Since the size of a garbled TM depends on the worst case run time τ of the TM, this results in a scheme with ciphertext size $O(\kappa, \tau, |M|)$, public and secret key size $O(\kappa, |M|)$ and hardness based on the existence of public key encryption.

We note that our construction of TMFE (Appendix D) compiles CktFE to TMFE, and by instantiating the underlying CktFE scheme with the PKE based CktFE construction of [27], we obtain the same parameters as above, i.e. ciphertext size $O(\kappa, \tau, |M|)$, public and secret key size $O(\kappa, |M|)$ and hardness based on the existence of public key encryption. However, by instantiating the CktFE scheme with a succinct FE scheme [18] we can shave off the machine size dependence on the public key and the ciphertext, resulting in shorter public key and ciphertext for TMs and optimal parameters for computation models such as DFAs. Note that for the case of DFAs, the public key and ciphertext size of the aforementioned construction depend on $|M|$, which our construction of DfaFE (Section 4) avoids. C.2. Additionally, even for the case of TMs, there are several applications where the machine size is very large and the runtime must be bounded: for instance, the machine may correspond to an automatic theorem prover [15] or a SAT solver of large size, and one may desire to obtain the best solution possible within a bounded amount of time.

We emphasize that converting the TM to a circuit and using the reusable garbled circuits construction [18] does not support dynamic data lengths, which is the main focus of this work.

Finally, we note that a sublcass of FE, namely, "attribute based encryption" for circuits, supporting unbounded length attributes has been constructed recently [11]. However, even in [11], the key generator and encryptor must agree on an input length ℓ and all inputs chosen by the encryptor must be of the same length ℓ in order to work with a given key for a circuit with input size ℓ. This is in contrast to our work where the input size for each message may be chosen dynamically.

B Preliminaries

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 from (and including) a to b. 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 $\mathrm{negl}(n)$ 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 $\mathrm{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$. The function $\log x$ is the base 2 logarithm of x.

B.1 Functional Encryption for Circuits

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

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.

- $\mathsf{CktFE.Setup}(1^\kappa)$ is a p.p.t. algorithm 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 p.p.t. algorithm that takes as input the master secret key MSK and a circuit $f \in \mathcal{F}_\kappa$ and outputs a corresponding secret key SK_f.
- $\mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x})$ is a p.p.t. algorithm that takes as input the master public key PK and an input message $\mathbf{x} \in \mathcal{X}_\kappa$ and outputs a ciphertext CT.
- $\mathsf{CktFE.Dec}(\mathsf{SK}_f, \mathsf{CT}_\mathbf{x})$ is 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 5** (Correctness)**.** A functional encryption scheme CktFE is correct if for all $f \in \mathcal{F}_\kappa$ and all $x \in \mathcal{X}_\kappa$,

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

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

▶ **Definition 6** (Compactness [1])**.** 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.

B.1.1 Simulation Security for Single Key Circuit FE.

In this section, we define simulation based security for single key Functional Encryption as in [18, Defn 2.13].

▶ **Definition 7** (FULL-SIM Security)**.** Let CktFE be a functional encryption scheme for a circuit family $\mathcal{F}$. For every p.p.t. adversary $A = (A_1, A_2)$ and a p.p.t. simulator Sim, consider the following two experiments:

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CktFE},A}(1^\kappa)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CktFE},\mathrm{Sim}}(1^\kappa)$:
1: $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{CktFE.Setup}(1^\kappa)$	1: $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{CktFE.Setup}(1^\kappa)$
2: $(f, st_1) \leftarrow A_1(\mathsf{PK})$	2: $(f, st_1) \leftarrow A_1(\mathsf{PK})$
3: $\mathsf{sk}_f \leftarrow \mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$	3: $\mathsf{sk}_f \leftarrow \mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$
4: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_f)$	4: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_f)$
5: $\mathsf{CT} \leftarrow \mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x})$	5: $\tilde{\mathsf{CT}} \leftarrow \mathrm{Sim}(\mathsf{PK}, \mathsf{sk}_f, f, f(\mathbf{x}), 1^{\lvert x\rvert})$
6: Output (st, CT)	6: Output $(st, \tilde{\mathsf{CT}})$

The functional encryption scheme CktFE is then said to be single query $\mathsf{FULL\text{-}SIM}$ secure if there exists a p.p.t. simulator Sim such that for every p.p.t. adversary $A = (A_1, A_2)$, the following two distributions are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CktFE},A}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CktFE},\mathrm{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

B.2 Turing Machines

We recall the definition of a Turing machine (TM). A TM M is represented by the tuple $(Q, \Gamma, \beta, \Sigma, \delta, q_{st}, F)$ where Q is a finite set of states, Γ is a finite alphabet, $\beta \in \Gamma$ is the blank symbol, $\Sigma \subseteq \Gamma \setminus \{\beta\}$ is the set of input symbols, q_{st} is the start state, $F = \{q_{acc}, q_{rej}\}$ where $q_{acc} \in Q$ is the accept state, $q_{rej} \in Q$ is the reject state and $\delta : Q \setminus F \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function (stored as a table). Upon input $\mathbf{w} = (w_1, \ldots, w_k) \in \Sigma^k$ for some arbitrary polynomial k, the machine M accepts the input if and only if given a tape initialised with the input $\mathbf{w}$ and the head at w_1, following the TM's transition function leads to q_{acc}. We say $M(\mathbf{w}) = 1$ iff M accepts $\mathbf{w}$ and 0 otherwise. We also denote the runtime (i.e. number of steps the head takes) by $\mathsf{runtime}(M, \mathbf{w})$.

B.2.1 Oblivious Turing Machines

Our construction makes use of oblivious Turing machines.

▶ **Definition 8** (Oblivious Turing Machine [22])**.** An Oblivious Turing Machine (OTM) is a Turing Machine for which there exists a function t such that, at every timestep i the machine head is at cell $t(i)$ regardless of the input.

Moreover there exist efficient transformations that convert any Turing machine M that takes time T to decide an input to an oblivious one that takes time $T \log T$ to decide the same input [25]. Here, we describe a simple transformation that incurs a quadratic blowup in running time.

Given a TM M, a simple OTM construction adds an additional marker for the head location. Now, to simulate step i in the TM, the OTM, scans from cell 1 to cell i, ensuring that it reads the current head location. Now, it moves back from cell i to 1, writing the correct symbol for the next step, while also updating the state. Once back at cell 1, simulation of step i is complete, and the OTM moves to a state simulate q_{i+1} and if q_{i+1} is not an accepting or rejecting state, it moves to simulating step $i + 1$. Since in step i, we would need to scan at most i cells (as that is the farthermost the head could have moved), an $O(t)$ computation, now takes $O(t^2)$. Also, if we are willing to reveal the runtime of the given input on the Turing Machine, then we can stop simulating after the last timestep t. A more efficient transformation due to Pippenger-Fischer[25] reduces the time required to $O(t \log t)$.

We note that a slightly different definition of OTMs [4] requires that the head movements are the same for *all* inputs of the same size, which would imply that the OTM runs in worst case time. However, if we are willing to reveal the running time of a machine on a given input, then the OTM can be made to halt once the input has been decided. In particular, if $\mathsf{runtime}(M_1(\mathbf{w}_1) = \mathsf{runtime}(M_2(\mathbf{w}_2)$, then the head movements of the OTM corresponding to M_1 and the OTM corresponding to M_2 are exactly the same.

B.3 Functional Encryption for Turing Machines

In this section, we will define functional encryption for Turing Machines (TM). The definition of Turing machines and oblivious Turing machines is recalled in Appendix B.2. We denote the runtime of Turing machine M on input $\mathbf{w}$ by $\mathsf{time}(M, \mathbf{w})$.

Let $\{\mathcal{M}_\kappa\}$ be a family of Turing machines with running time upper-bounded by a polynomial. 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.

- $\mathsf{TMFE.Setup}(1^\kappa)$ is a p.p.t. algorithm takes as input the unary representation of the security parameter and outputs the master public and secret keys $(\mathsf{PK}, \mathsf{MSK})$.
- $\mathsf{TMFE.KeyGen}(\mathsf{MSK}, \mathsf{M})$ is a p.p.t. algorithm that takes as input the master secret key MSK and a TM M and outputs a corresponding secret key SK_M.
- $\mathsf{TMFE.Enc}(\mathsf{PK}, \mathbf{w})$ is a p.p.t. algorithm that takes as input the master public key PK, and an input message $\mathbf{w}$, outputs a ciphertext $\mathsf{CT}_\mathbf{w}$.
- $\mathsf{TMFE.Dec}(\mathsf{SK}_\mathsf{M}, \mathsf{CT}_\mathbf{w})$ is a deterministic algorithm that takes as input the secret key SK_M and a ciphertext $\mathsf{CT}_\mathbf{w}$ and outputs a bit b.

▶ **Definition 9** (Correctness)**.** A functional encryption scheme TMFE is correct if for all $M \in \mathcal{M}$ and all $\mathbf{w} \in \Sigma^*$,

$$\Pr\left[\begin{array}{l}(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{TMFE.Setup}(1^\kappa);\\ \mathsf{TMFE.Dec}\Big(\mathsf{TMFE.KeyGen}(\mathsf{MSK}, \mathsf{M}), \mathsf{TMFE.Enc}(\mathsf{PK}, \mathbf{w})\Big) \neq \mathsf{M}(\mathbf{w})\end{array}\right] = \mathrm{negl}(\kappa)$$

where the probability is taken over the coins of $\mathsf{TMFE.Setup}$, $\mathsf{TMFE.KeyGen}$, ?and $\mathsf{TMFE.Enc}$.

B.3.1 Efficiency.

The efficiency property of a public-key FE scheme for Turing machines [2] says that the algorithm $\mathsf{TMFE.Setup}$ on input 1^κ should run in time polynomial in κ, $\mathsf{TMFE.KeyGen}$ on input the Turing machine M and the master key MSK should run in time polynomial in $(\kappa, |M|)$, $\mathsf{TMFE.Enc}$ on input a message $\mathbf{w}$ and the public key should run in time polynomial in $(\kappa, |\mathbf{w}|)$. Finally, $\mathsf{TMFE.Dec}$ on input a functional key of M and an encryption of $\mathbf{w}$ should run in time polynomial in $(\kappa, |M|, |\mathbf{w}|, \mathsf{time}(M, \mathbf{w}))$.

Our relaxation. We will achieve a relaxed version of efficiency which includes all the above conditions except ciphertext succinctness. We permit the size of the ciphertext to depend polynomially on (κ, τ) where τ is the maximum number of time steps the Turing machine may run on the input. Formally, for the family of Turing machines $\mathcal{M}$, let $T(n)$ be the polynomial upper bound on the running time for inputs of size n. Then $\tau = T(|\mathbf{w}|)$. To emphasize this dependence, we pass τ as an additional parameter to the encryption algorithm in the remainder of the paper.

B.3.2 Simulation Based Security for Single Key TM-FE.

In this section, we define simulation based security for single key FE for TMs. The definition is analogous to the definition of FE for circuits (Appendix B.1.1), except that the simulator additionally takes as input the worst case running time τ defined above and the actual running time of M on $\mathbf{x}$, i.e. $\mathsf{time}(M, \mathbf{x})$ which is revealed by the decryption operation.

▶ **Definition 10** (FULL-SIM- Security for TM-FE.)**.** Let $\mathcal{F}_\mathcal{M}$ be a functional encryption scheme for a TM family $\mathcal{M}$. For every p.p.t. adversary $A = (A_1, A_2)$ and a p.p.t. simulator Sim, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{TMFE},\mathsf{A}}(1^\kappa)}$**:**	$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{TMFE},\mathsf{Sim}}(1^\kappa)}$**:**
1: $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{TMFE.Setup}(1^\kappa)$	1: $(\mathsf{PK},\mathsf{MSK}) \leftarrow \mathsf{TMFE.Setup}(1^\kappa)$
2: $(M, st_1) \leftarrow A_1(\mathsf{PK})$	2: $(M, st_1) \leftarrow A_1(\mathsf{PK})$
3: $\mathsf{sk}_M \leftarrow \mathsf{TMFE.KeyGen}(\mathsf{MSK}, \mathsf{M})$	3: $\mathsf{sk}_M \leftarrow \mathsf{TMFE.KeyGen}(\mathsf{MSK}, \mathsf{M})$
4: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_M)$	4: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_M)$
5: $\mathsf{CT} \leftarrow \mathsf{TMFE.Enc}(\mathsf{PK}, \mathbf{x}, \tau)$	5: $\tilde{\mathsf{CT}} \leftarrow \mathrm{Sim}(\mathsf{PK}, \mathsf{sk}_M, M, M(\mathbf{x}), 1^{\lvert\mathbf{x}\rvert}, \tau, \mathsf{time}(M, \mathbf{x}))$
6: Output (st, CT)	6: Output $(st, \tilde{\mathsf{CT}})$

The TM functional encryption scheme is said to be single query FULL-SIM secure if there exists a p.p.t. simulator Sim such that for every p.p.t. adversary $A = (A_1, A_2)$, the following two distributions are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{TMFE},\mathsf{A}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{TMFE},\mathrm{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

B.4 Learning With Errors Assumption

The security of the single key FE for circuits construction of [18] is based on the hardness of the Learning With Errors (LWE) problem, introduced by Regev [26]. Solving the LWE problem on average is known to be at least as hard as solving approximate versions of certain standard lattice problems in the worst case [26, 10]. The best known algorithms for these problems for an approximation factor of 2^{n^ϵ} on n dimensional lattices run in time $2^{\tilde{O}(n^{1-\epsilon})}$ for any constant $0 < \epsilon < 1$. See [18] for a detailed discussion.

The LWE assumption is stated below.

▶ **Definition 11.** (Learning With Errors) Let q, α, m be functions of a parameter n. For a secret $\mathbf{s} \in \mathbb{Z}_q^n$, the distribution $A_{q,\alpha,\mathbf{s}}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is obtained by sampling $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and an $e \leftarrow D_{\mathbb{Z},\alpha q}$, and returning $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e) \in \mathbb{Z}_q^{n+1}$. The Learning With Errors problem $\mathsf{LWE}_{q,\alpha,m}$ is defined as follows: For $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, distinguish between the distributions:

$$D_0(\mathbf{s}) := \mathbf{U}(\mathbb{Z}_q^{m\times(n+1)}) \quad \text{and} \quad D_1(\mathbf{s}) := (A_{q,\alpha,\mathbf{s}})^m.$$

Here, $\mathbf{U}$ denotes the uniform distribution. The Learning With Errors assumption states that no p.p.t. algorithm $\mathcal{A}$ can distinguish between $D_0(\mathbf{s})$ and $D_1(\mathbf{s})$ with non-negligible advantage (over the random coins of $\mathcal{A}$ and the randomness of the samples).

Intuitively, subexponential hardness of LWE assumes that achieving an approximation factor of 2^{n^ϵ} in the underlying lattice problem is hard for a polynomial time algorithm.

C Decomposable Functional Encryption for Circuits

In this section, we establish that the succinct single key FE for circuits by Goldwasser et al. [18] and the non-succinct functional encryption scheme by Sahai and Seyalioglu [27] are both decomposable.

C.1 Decomposable Functional Encryption for Circuits from LWE

▶ **Lemma 12.** *The single key, succinct functional encryption scheme for circuits by Goldwasser et al. [18] is decomposable.*

Proof. For completeness, we recall the details of the GKPVZ scheme. Since decomposability depends only on the structure of the public key and the ciphertext, we omit the details of key generation and decryption. We refer the interested reader to [18] for more details.

The GKPVZ scheme uses as building blocks the following primitives:

- A fully homomorphic encryption scheme. We denote the length of ciphertexts produced by this scheme, both by encryption and by evaluation, as λ. We instantiate the fully homomorphic scheme by the construction of Brakerski, Gentry and Vaikuntanathan [9], and refer to this as FHE.
- A single key attribute based encryption scheme for circuits [2]. We instantiate the ABE scheme by the construction of Gorbunov, Vaikuntanathan and Wee [19] and refer to it as ABE. The ABE scheme of Gorbunov et al. builds upon a primitive which they call "two-one-recoding" and which they instantiate from the LWE assumption. We will denote this construction by TOR.
- A garbled circuit. We instantiate the garbled circuit by Yao's classic construction [30] and refer to it as Yao.

$\mathsf{FE.Setup}(1^\kappa, 1^\ell)$: Upon input the security parameter κ and the length of the input ℓ, do the following.

1. Define $\ell_1 = |\mathsf{HPK}| + \ell \cdot \lambda$, where HPK denotes the FHE public key, and λ denotes the length of an FHE ciphertext. Now, run the $\mathsf{ABE.Setup}(1^\kappa, 1^{\ell_1})$ algorithm λ times and obtain:
$$\left(\mathsf{FMPK}_j, \mathsf{FMSK}_j \right) \leftarrow \mathsf{ABE.Setup}(1^\kappa, 1^{\ell_1}) \quad \forall\, j \in [\lambda]$$
2. By the construction of [19, Sec 6.1], we additionally have for each $j \in [\lambda]$:
$$\mathsf{FMPK}_j = \left(\{\mathsf{TOR.pk}_{j,k,b}\}_{k \in [\ell_1],\, b \in \{0,1\}}, \mathsf{TOR.pk}_{j,\mathsf{out}} \right)$$
3. Partition the interval $[\ell_1]$ into $\ell + 1$ contiguous, disjoint intervals: the first interval S_0 of size $|\mathsf{HPK}|$ and ℓ intervals $S_1, \ldots, S_\ell$ of size λ each. Formally,
   a. Define $S_0 = [1,\ |\mathsf{HPK}|\ ]$.
   b. Initialize $k_1 = |\mathsf{HPK}|$, $k_2 = k_1 + \lambda$ and $i = 1$. Then, while $i < \ell$,
      - Define $S_i = [k_1 + 1,\ k_2]$.
      - Update k_1 as $k_1 = k_2$. Increment i.
4. Next, for each $j \in [\lambda]$, partition the $2 \cdot \ell_1$ TOR public keys according to the above sets:
$$\forall\, i \in [0, \ell],\ \ \widetilde{\mathsf{TOR}}.\mathsf{pk}_{j,i} \leftarrow \left\{\mathsf{TOR.pk}_{j,k,b}\right\}_{k \in S_i,\, b \in \{0,1\}}$$
   Further, we define:
$$\forall\, i \in [0, \ell],\ \ \widetilde{\mathsf{TOR}}.\mathsf{pk}_i \leftarrow \left\{\widetilde{\mathsf{TOR}}.\mathsf{pk}_{j,i}\right\}_{j \in [\lambda]} \quad \text{and} \quad \widetilde{\mathsf{TOR}}.\mathsf{pk}_{\mathsf{out}} \leftarrow \{\mathsf{TOR.pk}_{j,\mathsf{out}}\}_{j \in [\lambda]}$$
5. Thus, we may write for $i \in [1, \ell]$:
$$\mathsf{PK}_i = \widetilde{\mathsf{TOR}}.\mathsf{pk}_i, \quad \mathsf{PK}_{\mathsf{indpt}} = \left(\widetilde{\mathsf{TOR}}.\mathsf{pk}_0,\ \widetilde{\mathsf{TOR}}.\mathsf{pk}_{\mathsf{out}}\right)$$

[2] Although technically the GKPVZ scheme uses a two-outcome ABE scheme which is slightly different from a single outcome ABE scheme, we will ignore this technicality for ease of notation here. None of the arguments are affected by this minor omission as the two-outcome ABE scheme essentially uses two single outcome ABE scheme ciphertexts.

$\mathsf{FE.Enc}(\mathsf{PK}, \mathbf{x})$: Upon input, the public key and the input vector $\mathbf{x}$ where $|\mathbf{x}| = \ell$, do the following:

1. Generate a fresh key pair $(\mathsf{HPK}, \mathsf{HSK}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$ for the FHE scheme. Encrypt each bit x_i as $\phi_i = \mathsf{FHE.Enc}(\mathsf{HPK}, x_i)$. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_\ell)$.
2. Run Yao's garbled circuit algorithm to produce a garbled circuit for the FHE decryption algorithm. Note that the number of input wires for this circuit is λ. Let the labels of the garbled circuit be denoted by $\mathbf{L}_j = (L_{j,0}, L_{j,1})$ for $j \in [\lambda]$. Let the garbled circuit itself be denoted by Γ.
3. For $j \in [\lambda]$, let
$$\boldsymbol{\psi}_j \leftarrow \mathsf{ABE}_j.\mathsf{Enc}(\mathsf{FMPK}_j, \mathbf{L}_j, \mathsf{attr}_j)$$
Here, the message is $\mathbf{L}_j$ and the public attribute is $\mathsf{attr}_j = \mathsf{attr} = (\mathsf{HPK}, \phi)$. Note that $|\mathsf{attr}| = \ell_1 = |\mathsf{HPK}| + \ell \cdot \lambda$, since $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_\ell)$ and each $|\phi_i| = \lambda$. We analyze the structure of $\boldsymbol{\psi}_j$ below.
   a. By the description of the ABE scheme, $\boldsymbol{\psi}_j = (\psi_{j,1}, \ldots, \psi_{j,\ell_1}, \tau_j)$ where:
$$\psi_{j,k} \leftarrow \mathsf{TOR.Encode}\big(\mathsf{TOR.pk}_{j,k,\mathsf{attr}_k}(s)\big), \quad \forall\, k \in [\ell_1]$$
$$\tau_j \leftarrow \mathsf{TOR.E}\big(\mathsf{TOR.Encode}(\mathsf{TOR.pk}_{j,\mathsf{out}}, s), \mathbf{L}_j\big)$$
   Here $s \leftarrow \mathcal{S}$, where $\mathcal{S}$ is a set defined in the ABE construction [3].
   b. For $j \in [\lambda]$, partition the ciphertexts $\{\psi_{j,k}\}_{k \in [\ell_1]}$ according to the sets $S_0, \ldots, S_\ell$ as:
$$\widetilde{\boldsymbol{\psi}}_{j,i} = \{\psi_{j,k}\}_{k \in S_i} \quad \text{for} \quad j \in [\lambda],\ i \in [0, \ell]$$
   Further, we define:
$$\widetilde{\boldsymbol{\psi}}_i = (\boldsymbol{\psi}_{j,i}) \quad \text{for} \quad j \in [\lambda],\ i \in [0, \ell] \quad \text{and} \quad \boldsymbol{\tau} = (\tau_1, \ldots, \tau_\lambda)$$
4. For $i \in [\ell]$, we define $\mathsf{CT}_i = (\phi_i, \widetilde{\boldsymbol{\psi}}_i)$, $\mathsf{CT}_{\mathsf{indpt}} = (\widetilde{\boldsymbol{\psi}}_0, \boldsymbol{\tau}, \Gamma)$

Thus, we have established that the public key and the ciphertext of the GKPVZ scheme have the structure of a decomposable FE scheme. Next, we discuss the instantiations of common and independent randomness required by decomposable FE in GKPVZ.

Common and Independent Randomness in GKPVZ. In the GKPVZ scheme, the role of common randomness is played by the FHE public key HPK, and the input $s \in \mathcal{S}$ to $\mathsf{TOR.Encode}$. The remaining randomness used in the GKPVZ encryption, such as the randomness chosen by $\mathsf{FHE.Enc}$ and $\mathsf{TOR.Encode}$ is independent randomness that may be chosen independently by each invocation of $\mathcal{E}$.

Hence, we have established that the GKPVZ scheme [18] is decomposable. ◀

C.2 Decomposable Functional Encryption for Circuits from PKE

In this section, we show that the single key non-succinct functional encryption scheme by Sahai and Seyalioglu [27] is decomposable.

We recall the construction here. Let $\mathcal{F} : \mathcal{X} \to \{0,1\}$ be the function family and let N be the bit size of a function $f \in \mathcal{F}$. Additionally, let n be the bit size of $\mathbf{x} \in \mathcal{X}$. We will

[3] The precise choice of $\mathcal{S}$ is not important for us, it suffices to observe that $\mathcal{S}$ is chosen as a function of the security parameter independent of anything else. For readers familiar with the [19] construction, $\mathcal{S}$ is the set $\mathbb{Z}_q^n$ from which the LWE secret s is sampled.

make use of PKE, which is a standard IND-CPA public key encryption scheme. Let U be the universal circuit so that $U(\mathbf{x}, f) = f(\mathbf{x})$.

$\mathsf{CktFE.Setup}(1^\kappa)$: Sample $2N$ PKE keys $(\mathsf{PK}_{i,b}, \mathsf{SK}_{i,b}) \leftarrow \mathsf{PKE.Setup}(1^\kappa)$ where $i \in [N]$, $b \in \{0,1\}$. Output public key $\mathsf{PK} = \{\mathsf{PK}_{i,b}\}$ and the master key $\mathsf{MSK} = \{\mathsf{SK}_{i,b}\}$ for $i \in [N]$, $b \in \{0,1\}$.

$\mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x})$: Upon input the public key and input $\mathbf{x}$, do the following:

1. Construct a Yao's garbled circuit Γ for U, and let $\{L_{i,b}\}_{i\in[n],b\in\{0,1\}}$ be the labels corresponding to the first input and $\{L'_{i,b}\}_{i\in[n+1,n+N],b\in\{0,1\}}$ be the labels corresponding to the second input.
2. Let $\hat{\mathbf{L}}_{i,b} = \mathsf{PKE.Enc}(L'_{i,b}, \mathsf{PK}_{i,b})$.
3. Sample $b_i \leftarrow \{0,1\}$ for $i \in [n+1, n+N]$
4. Output $\mathsf{CT}_\mathbf{x} = \left(\Gamma, \{\{L_{i,x_i}\}_{i\in[n]}\}, \{\hat{\mathbf{L}}_{i,b_i}, \hat{\mathbf{L}}_{i,\bar{b}_i}\}_{i\in[n+1,n+N],b_i\in\{0,1\}}\right)$

$\mathsf{CktFE.KeyGen}(\mathsf{MSK}, \mathsf{f})$: Let $f_1 \ldots . f_N$ be the bits corresponding to f. Output as SK_f the secret keys corresponding to the bits of f, namely SK_{i,f_i} for $i \in [N]$.

$\mathsf{CktFE.Dec}(\mathsf{SK_f}, \mathsf{CT})$: Given the secret keys SK_{i,f_i}, try to decrypt both $(\hat{\mathbf{L}}_{i,b_i}, \hat{\mathbf{L}}_{i,\bar{b}_i})$ for $i \in [n+1, n+N]$. Exactly one will decrypt to yield the label L_{i,f_i}. Evaluate the garbled circuit using labels $(L_{1,x_1}, \ldots, L_{n,x_n}, L'_{1,f_1}, \ldots, L'_{N,f_N}$ for $\mathbf{x}$ and f to get the output bit corresponding to $f(\mathbf{x})$.

Decomposability follows trivially : we may view the garbled circuit Γ and the encrypted labels $\{\hat{\mathbf{L}}_{i,b_i}, \hat{\mathbf{L}}_{i,\bar{b}_i}\}_{i\in[n+1,n+N],b_i\in\{0,1\}}\big)$ as $\mathsf{CT}_{\mathsf{indpt}}$, and the labels L_{i,x_i} corresponding to x_i as the input dependent ciphertext components CT_i for $i \in [n]$. We note that the input dependent part of the ciphertext is succinct.

D Single Key Functional Encryption for Turing Machines

In this section, we will construct a single key (public key) functional encryption scheme for TMs. We decompose the input into three components of size ℓ_1, ℓ_2 and ℓ_3 each, where the second and third components are further decomposed bit by bit. We will use the first component to encrypt the symmetric keys for selecting the next state and the tape symbol to be written at the current location, the second component is for the current state in the TM computation, and the third component is for the current tape symbol to be read.

D.1 Construction

Let $\mathcal{M}_\kappa$ be a TM family. Let Q be the size of the state space of the TM family and Γ be the size of the alphabet. Then, we define:

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

1. Choose a symmetric key encryption scheme SKE with key space $\mathcal{K}$.
2. Define a circuit family as follows. Let $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} = (\mathcal{K}^{2\log Q} \times \mathcal{K}^{2\log \Gamma}) \times Q \times \Gamma$ and $\mathcal{Y} = (\mathcal{K}^{\log Q} \times \mathcal{K}^{\log \Gamma})$. We set

$$\ell_1 = \lfloor \mathcal{K}^{2\log Q} \rfloor + \lfloor \mathcal{K}^{2\log \Gamma} \rfloor, \quad \ell_2 = \lfloor Q \rfloor, \quad \ell_3 = \lfloor \Gamma \rfloor$$

where $\lfloor \cdot \rfloor$ denotes size in bits. Let $\ell = \ell_1 + \ell_2 + \ell_3$.

3. Invoke $\mathsf{CktFE.Setup}(1^\kappa, 1^\ell)$ to obtain

$$\mathsf{PK} = \big(\mathsf{PK}_1, (\mathsf{PK}_{2,1}, \ldots, \mathsf{PK}_{2,\log Q}), (\mathsf{PK}_{3,1}, \ldots, \mathsf{PK}_{3,\log \Gamma}), \mathsf{PK}_{\mathsf{indpt}}\big), \quad \mathsf{MSK}$$

4. Output $(\mathsf{PK}, \mathsf{MSK})$.

$\mathsf{TMFE.Enc}(\mathsf{PK}, \mathbf{w}, \tau)$: Upon input the public key PK, the message $\mathbf{w}$ of arbitrary length k and a bound τ on the runtime of the Turing machine, do the following.

1. **Sample the required common randomness and SKE keys**:

   a. Sample "common" randomness $r_i \leftarrow \mathcal{R}_1$ for $i \in [\tau]$.
   b. Sample SKE keys as follows. We define

$$\mathbf{K}_{i+1} = \Big((K^0_{(i+1,1)}, K^1_{(i+1,1)}), \ldots, (K^0_{(i+1,\log Q)}, K^1_{(i+1,\log Q)})\Big)$$

$$\mathbf{K}'_{i+1} = \Big((K'^0_{(i+1,1)}, K'^1_{(i+1,1)}), \ldots, (K'^0_{(i+1,\log \Gamma)}, K'^1_{(i+1,\log \Gamma)})\Big)$$

   where $K^b_{i+1,j} \leftarrow \mathsf{SKE.KeyGen}(1^\kappa)$ for $i \in [\tau - 1]$, $b \in \{0,1\}$, $j \in [\log Q]$ and $K'^b_{i+1,j} \leftarrow \mathsf{SKE.KeyGen}(1^\kappa)$ for $i \in [\tau - 1]$, $b \in \{0,1\}$, $j \in [\log \Gamma]$.
   c. Let $t : [\tau] \to [\tau]$ be the function [4] where $t(i)$ gives the time step in which the symbol being written in step i will be next read by an oblivious TM. Also let $t' : [\tau] \to \{0,1\}$ be a function such that $t'(i)$ is 0 if the symbol being read in timestep i is being read for the first time. Let $T = \{i : i \in [k+1, \tau], t'(i) = 0\}$
   d. Define message $\mathbf{y}_i = (\mathbf{K}_{i+1}, \mathbf{K}'_{t(i)})$ for $i \in [\tau]$.

2. **Compute CktFE ciphertexts τ steps**:

   a. For $i \in [\tau]$, we define:

$$\mathbf{c}_{i,1} = \bar{\mathcal{E}}(\mathsf{PK}_1, \mathbf{y}_i, r_i), \quad \mathbf{c}_{i,2} = \bar{\mathcal{E}}(\mathsf{PK}_{\mathsf{indpt}}, r_i), \quad \mathbf{c}_i = (\mathbf{c}_{i,1}, \mathbf{c}_{i,2})$$

   Thus $\mathbf{c}_i$ is the CktFE ciphertext component in the i^{th} step that encodes that encodes the SKE secret keys required to "select" the next state and tape symbol to be written.

   b. Let $\mathbf{d}_1 = \bar{\mathcal{E}}\ (\mathsf{PK}_2, q_{st}, r_1)$. Here q_{st} denotes the start state of the TM. Further, let

$$\mathbf{d}^b_{i,j} = \mathcal{E}\ (\mathsf{PK}_{2,j}, b, r_i)\ \ \forall\ i \in [1,\tau],\ j\ \in [\log Q],\ b \in \{0,1\}.$$
$$\mathbf{d}_{i,j} \triangleq \big(\mathbf{d}^0_{i,j}, \mathbf{d}^1_{i,j}\big)$$
$$\mathbf{d}_{i,q} \triangleq (\mathbf{d}^{q_j}_{i,j})\ \ \forall j \in [\log Q] \text{ where } q_j \text{ is the } j\text{th bit of } q.$$

   Thus, $\mathbf{d}_{i,j}$ is the CktFE ciphertext component that encodes both possibilities for the j^{th} bit of the state obtained in the i^{th} step. $\mathbf{d}_{i,q}$ denotes the CktFE ciphertext component obtained by "selecting" the bits corresponding to state q at step i.

[4] implied by an oblivious TM which runs for τ steps

c. Let $\mathbf{e}_i = \bar{\mathcal{E}}\ (\mathsf{PK}_3, \mathbf{w}_i, r_i)$ for $i \in [1,k]$ and $\mathbf{e}_i = \bar{\mathcal{E}}\ (\mathsf{PK}_3, \beta, r_i)$ where β is the blank symbol, for all $i \in T$. Further, let

$$\mathbf{e}^b_{i,j} = \mathcal{E}\ (\mathsf{PK}_{3,j}, b, r_i)\ \ \forall\, i \in [1,\tau]\ j\ \in [\log \Gamma],\ b \in \{0,1\}.$$
$$\mathbf{e}_{i,j} \triangleq \left(\mathbf{e}^0_{i,j}, \mathbf{e}^1_{i,j}\right)$$
$$\mathbf{e}_{i,\gamma} \triangleq (\mathbf{e}^{\gamma_j}_{i,j})\ \ \forall j \in [\log \Gamma] \text{ where } \gamma_j \text{ is the } j\text{th bit of } \gamma.$$

Thus, $\mathbf{e}_{i,j}$ is the CktFE ciphertext component that encodes both possibilities for the j^{th} bit of the tape symbol read in the i^{th} step. $\mathbf{e}_{i,\gamma}$ denotes the CktFE ciphertext component obtained by "selecting" the bits corresponding to symbol γ at step i.

3. Compute SKE encryptions of CktFE ciphertexts:

a. For $i \in [1,\tau]$, $j \in [\log Q]$, $b \in \{0,1\}$ encrypt each $\mathbf{d}^b_{i,j}$ using the corresponding SKE key $K^b_{i,j}$ as:

$$\hat{\mathbf{d}}^b_{i,j} = \mathsf{SKE.Enc}(K^b_{i,j}, \mathbf{d}^b_{i,j})$$

b. Similarly, for $i \in [1,\tau]$, $j \in [\log \Gamma]$, $b \in \{0,1\}$ encrypt each $\mathbf{e}^b_{i,j}$ using the corresponding SKE key $K'^b_{i,j}$ as:

$$\hat{\mathbf{e}}^b_{i,j} = \mathsf{SKE.Enc}(K'^b_{i,j}, \mathbf{e}^b_{i,j})$$

4. Shuffle SKE encryptions in every position:

a. Choose $b_{i,j} \leftarrow \{0,1\}$ randomly for $i \in [1,\tau]$, $j \in [\log Q]$ and define:

$$\hat{\mathbf{D}}_{i,j} = \left(\ \hat{\mathbf{d}}^{b_{i,j}}_{i,j}, \hat{\mathbf{d}}^{\bar{b}_{i,j}}_{i,j}\ \right), \quad \hat{\mathbf{D}}_i = (\hat{\mathbf{D}}_{i,j})$$

b. Choose $b_{i,j} \leftarrow \{0,1\}$ randomly for $i \in [1,\tau]$, $j \in [\log \Gamma]$ and define:

$$\hat{\mathbf{E}}_{i,j} = \left(\ \hat{\mathbf{e}}^{b_{i,j}}_{i,j}, \hat{\mathbf{e}}^{\bar{b}_{i,j}}_{i,j}\ \right), \quad \hat{\mathbf{E}}_i = (\hat{\mathbf{E}}_{i,j})$$

5. Output the ciphertext:

$$\mathsf{CT}_{\mathbf{w}} = \left(\{\mathbf{c}_i, \hat{\mathbf{D}}_i, \hat{\mathbf{E}}_i\}\ \ \forall\, i \in [\tau], \quad \mathbf{d}_1, \{\mathbf{e}_i\}\ \ \forall\ \ i \in [1,k] \cup T\right)$$

$\mathsf{TMFE.KeyGen}(\mathsf{MSK}, \mathsf{M})$: Let M denote a TM machine. Transform M into an oblivious Turing machine M_O by applying the Pippenger-Fischer transformation[25] and T denote M_O's transition matrix. Let T_i denote the i^{th} row of T, with format $\left((\gamma, q) \to (\gamma', q')\right)$ indicating that the machine enters state q' upon input symbol γ and input state q, and writes γ' to it's current position. Let $\mathsf{SK}_M = \mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$ where f is defined in Figure 2.

$\mathsf{TMFE.Dec}(\mathsf{SK}_M, \mathsf{CT}_{\mathbf{w}})$: Parse ciphertext $\mathsf{CT}_{\mathbf{w}} = \left(\ \mathbf{c}_i, \hat{\mathbf{D}}_i, \hat{\mathbf{E}}_i)_{i\in[1,\tau]}, \mathbf{d}_1, (\mathbf{e}_i)_{i\in[1,k]\cup T}\right)$ For $i \in [1,k] \cup T$, let $\mathbf{d}_{1,q_1} = \mathbf{d}_1$ and $\mathbf{e}_{i,\gamma_i} = \mathbf{e}_i$.
Initialize $i = 1$. While computation does not output 0 or 1, do the following:

- Let $\mathsf{CT}_i = (\mathbf{c}_i, \mathbf{d}_{i,q_i}, \mathbf{e}_{i,\gamma_i})$. Recall that $\mathbf{d}_{i,q_i} = (\mathbf{d}^{q_{i,j}}_{i,j})$ for $j \in [\log Q]$ and $\mathbf{e}_{i,\gamma_i} = (\mathbf{d}^{\gamma_{i,j}}_{i,j})$ for $j \in [\log \Gamma]$.
- Let $\left((K_{i+1,1}, \ldots, K_{i+1,\log Q}), (K'_{t(i),1}, \ldots, K'_{t(i),\log \Gamma})\right) = \mathsf{CktFE.Dec}(\mathsf{SK}_f, \mathsf{CT}_i)$.

Function $f(\mathbf{K}, \mathbf{K}', q, \gamma)$

1. Lookup the transition table T for (q, γ). Say that $(q, \sigma) \rightarrow (q', \gamma')$. If no entry is found, output $\perp$ and exit.
2. Check if q' is q_{acc} or q_{rej}. If the former, output 1, else output 0.
3. If q' is not an accepting state, parse $\mathbf{K}$ as $\{(K_j^0, K_j^1)\}$ for $j \in [\log Q]$, $b \in \{0,1\}$, and $\mathbf{K}'$ as $\{(K_j'^0, K_j'^1)\}$ for $j \in [\log \Gamma]$, $b \in \{0,1\}$. Choose the $\log Q$ keys $K_j^{q'_j}$ (for $j \in [\log Q]$), corresponding to the bits of q'. Similarly, choose the $\log \Gamma$ keys $K'^{\gamma'_j}_j$ (for $j \in [\log \Gamma]$), corresponding to the bits of γ' and output both these sets.

Figure 2 Function to provide keys for next state in TM computation.

- For $j \in [\log Q]$, try to decrypt each value in $\hat{\mathbf{D}}_{i+1,j}$ using obtained key $K_{i+1,j}$. Exactly one of the two ciphertexts per bit position will be decrypted, say $\hat{\mathbf{d}}^{b_j}_{i+1,j}$. Set

$$\mathbf{d}_{i+1,q_{i+1}} = \left(\mathsf{SKE.Dec}(K_{i+1,j}, \hat{\mathbf{d}}^{b_j}_{i+1,j}) \right) \quad \forall j \in [\log Q]$$

- Analogously, for $j \in [\log \Gamma]$, try to decrypt each value in $\hat{\mathbf{E}}_{t(i),j}$ using obtained key $K'_{t(i),j}$. Exactly one of the two ciphertexts per bit position will be decrypted, say $\hat{\mathbf{e}}^{b_j}_{t(i),j}$. Set

$$\mathbf{e}_{t(i),\gamma_{t(i)}} = \left(\mathsf{SKE.Dec}(K'_{t(i),j}, \hat{\mathbf{e}}^{b_j}_{t(i),j}) \right) \quad \forall j \in [\log \Gamma]$$

- Increment i.

D.1.1 Correctness and Efficiency.

Correctness. Note that in the encryption, the first component $\mathbf{c}_i$ encrypts message $\mathbf{y}_i$, which contains the set of all $2 \log Q$ symmetric keys used to construct SKE encryptions of the $(i+1)^{th}$ state and the $2 \log \Gamma$ keys used to construct SKE encryptions of the $t(i)^{th}$ input symbol. In the second component, the element $\mathbf{d}^b_{i,j}$ in tuple ($\mathbf{d}^b_{i,j}$) for $j \in [\log Q]$ and $b \in \{0,1\}$, contains an encryption of bit b, corresponding to the event that the j^{th} bit of i^{th} state is b. The set $\hat{\mathbf{D}}_i$ contains $2 \log Q$ SKE encryptions of $\mathbf{d}^b_{i,j}$ under keys $K^b_{i,j}$, shuffled for each position j. $\hat{\mathbf{E}}_i$ is constructed similarly but for the symbol that is to be written at that step. Decryption at step $i-1$ provides the level i symmetric keys $K^{b_j}_{i,j}$ to unlock the $\mathbf{d}^{b_j}_{i,j}$ for the correct next state of the computation q', i.e. $b_j = q'_{i,j}$ and the analogous keys for the symbol to be read at time $t(i)$. Thus, the decryptor recovers exactly the components $\mathbf{d}^{b_j}_{i,j}$ which may be combined to create the ciphertext $\mathbf{d}_{i,q_i}$. Along with this, before a symbol at time i is read, the corresponding $\mathbf{e}_i$ has been created, since the keys for step i were generated at time $j < i$ s.t. $t(j) = i$. Put together with $\mathbf{c}_i$ this yields an encryption of $(\mathbf{K}_{i+1}, \mathbf{K}'_{t(i)}, q_i, \gamma_i)$ which may now be decrypted with the function key to obtain the appropriate symmetric key encryption keys to decrypt the correct $\mathbf{d}_{i+1,q_{i+1}}$ and $\mathbf{e}_{t(i),\gamma_{t(i)}}$, thus propagating the computation.

Next, we provide the formal argument that the FE scheme provided in Section D is correct.

Formally, let k denote the length of input $\mathbf{w}$, let $q_1, \ldots, q_\tau$ denote the states visited by the TM during computation (we assume that there are τ steps but the same arguments work

for any number of steps $\leq \tau$). We have by correctness of decomposable functional encryption that:

$\forall i \in [\tau],\ \mathsf{CktFE.Enc}\big(\ \mathsf{PK}, (\mathbf{K}_{i+1}, \mathbf{K}'_{t(i)}, q_i, \gamma_i)\ \big) = (\mathbf{c}_i, \mathbf{d}_{i,q_i}, \mathbf{e}_{i,\gamma_i})$ where

$\mathbf{c}_i = \Big(\ \bar{\mathcal{E}}\big(\mathsf{PK}_1, \mathbf{y}_i, r_i\big),\ \bar{\mathcal{E}}\big(\mathsf{PK}_{\mathsf{indpt}}, r_i\big)\Big),\quad \mathbf{d}_{i,q_i} = \big(\mathcal{E}(\mathsf{PK}_{2,j}, q_{i,j}, r_i)\big)_{j\in[\log Q]}$

$\mathbf{e}_{i,\gamma_i} = \big(\mathcal{E}(\mathsf{PK}_{3,j}, \gamma_{i,j}, r_i)\big)_{j\in[\log \Gamma]}$

We have that, $\mathsf{CktFE.Dec}\big((\mathbf{c}_i, \mathbf{d}_{i,q_i}, \mathbf{e}_{i,\gamma_i}), \mathsf{SK}_f\big) = (\mathbf{K}_{q_{i+1}}, \mathbf{K}'_{t(i)})$ s.t.

$\mathbf{K}_{q_{i+1}} \triangleq (K^{b_1}_{i+1,1}, \ldots, K^{b_{\log Q}}_{i+1,\log Q})$ where $b_j = q_{i+1,j}$.

$\mathbf{K}'_{t(i)} \triangleq (K'^{b_1}_{t(i),1}, \ldots, K'^{b_{\log \Gamma}}_{t(i),\log \Gamma})$ where $b_j = \gamma_{t(i),j}$.

Now, we use the obtained keys to perform SKE decryption on $\hat{\mathbf{D}}_{i+1}$ and recover the CktFE ciphertexts corresponding to state q_{i+1}. For $j \in [\log Q]$, both elements of $\hat{\mathbf{D}}_{i+1,j}$ are attempted for decryption by $K^{b_j}_{i+1,j}$, of which only the element encoding the correct bit $q_{i+1,j}$ is recovered. Formally, we have:

$\hat{\mathbf{D}}_{i+1,j} = \big(\ \hat{\mathbf{d}}^0_{i+1,j}, \hat{\mathbf{d}}^1_{i+1,j}\ \big)$ and

$\mathsf{SKE.Dec}\ \big(\ K^{b_j}_{i+1,j},\ \hat{\mathbf{d}}^b_{i+1,j}\ \big) = \perp$ if $b_j \neq b$, and $\mathbf{d}^{q_{i+1,j}}_{i+1,j}$ otherwise.

By putting together all the components, we get by decomposability:

$$\mathbf{d}_{i+1,q_{i+1}} = \big(\ \mathbf{d}^{q_{i+1,j}}_{i+1,j}\ \big)\ \ \forall\, j \in [\log Q]$$

Using the same argument we get that that for $(i+1) \notin T$

$$\mathbf{e}_{i+1,\gamma_{i+1}} = \big(\ \mathbf{e}^{\gamma_{i+1,j}}_{i+1,j}\ \big)\ \ \forall\, j \in [\log \Gamma]$$

while for $(i+1) \in T, \mathbf{e}_{i+1,\gamma_{i+1}}$ is provided directly in the ciphertext, without SKE encryption. Now, since each component of $\mathbf{d}_{i+1,q_{i+1}}$ and $\mathbf{e}_{i+1,q_{i+1}}$ uses the same common randomness r_{i+1} as is used by $\mathbf{c}_{i+1}$, we have that $\mathsf{CT}_{i+1} = (\mathbf{c}_{i+1}, \mathbf{d}_{i+1,q_{i+1}}, \mathbf{e}_{i+1,\gamma_{i+1}})$, hence we may repeat while $i \leq \tau$. Here, we have used the fact that when we arrive at timestep i, either $i \in T$ or $\exists j < i$ s.t. $t(j) = i$ which follows from the construction of t. Finally for $i = \tau$,

$\mathsf{CktFE.Enc}\big(\ \mathsf{PK}, (\mathbf{c}_\tau, q_\tau, \gamma_\tau)\ \big) = (\mathbf{c}_\tau, \mathbf{d}_{\tau,q_\tau}, \mathbf{e}_{\tau,q_\tau})$

$\mathsf{CktFE.Dec}\big((\mathbf{c}_\tau, \mathbf{d}_{\tau,q_\tau}, \mathbf{e}_{\tau,q_\tau}), \mathsf{SK}_f\big) = 1$ iff $q_{\tau+1}$ is an accepting state, 0 otherwise.

Efficiency.

We note that our public key is short, i.e. polynomial in κ, since this is a CktFE public key meant to encode a fixed length message comprising SKE keys, state and tape symbol, and is independent of the size of the message $\mathbf{w}$ being encrypted or the machine size M in TMFE. Our function key for a TM M is a CktFE function key for the circuit defined in Figure 2, whose size is proportional to the size of M. Decryption of $\mathsf{CT}_{\mathbf{w}}$ and SK_M takes time proportional to $\mathsf{runtime}(M(\mathbf{w}))$, since we invoke $\mathsf{CktFE.Dec}$ for each step of the TM computation. We note that the trivial approach of converting the TM M into a circuit and invoking CktFE to compute $M(\mathbf{w})$ directly would force the circuit to have size proportional to the worst case running time of M. Thus our decryption time is of the order $\mathsf{runtime}(M(\mathbf{w})) \cdot \mathsf{Time}(\mathsf{CktFE.Dec})$, where $\mathsf{CktFE.Dec}$ takes time polynomial in $(\kappa, \ell, |M|)$ (we refer the reader to the description of $\mathsf{TMFE.Setup}(1^\kappa)$ in Section D.1 to check that $\ell = \mathrm{poly}(\kappa)$). This results in total decryption time polynomial in $(\kappa, |M|, \mathsf{runtime}(M, \mathbf{w}))$.

We now analyze the ciphertext size. Recall that τ is the worst case running time of the TM on a given input (see Appendix B.2 for the formal definition). Since we rely on succinct CktFE [18], we have TMFE ciphertext size $\mathrm{poly}(\kappa, \tau)$.

Machine Privacy. As in the case of DFA (Appendix E), our construction for Turing machines may be compiled with symmetric key encryption to achieve machine privacy.

D.2 Proof of Security

We proceed to show that our construction is secure. The proof of security is similar to the DFA case (Section 4.2), in that the CktFE simulator is invoked repeatedly to produce the CktFE ciphertexts. The CktFE ciphertexts are then encrypted using SKE to produce the TMFE ciphertext. However, there are two primary differences from the DFA case. First, the CktFE ciphertext now contains 3 components which encode the SKE secret keys (in component CT_1), the state (in component CT_2) and the work tape symbol (in component CT_3). The TMFE ciphertext contains CT_1 as well as SKE encryptions of CT_2 and CT_3, and the correct SKE keys are "selected" by the CktFE decrypt operation as we step through the computation. Second and more importantly, even though the computation ends at step $\mathsf{runtime}(M, \mathbf{w})$, we must simulate ciphertexts until time step τ, i.e. the worst case run time of the TM. After $\mathsf{runtime}(M, \mathbf{w})$ steps, the computation halts, and no longer produces any output. Thus, it is unclear how to invoke the CktFE simulator to produce ciphertexts for time steps $i \in [\mathsf{runtime}(M, \mathbf{w}) + 1, \ldots, \tau]$, although the TMFE simulator must produce τ ciphertexts.

The first difference can be handled by extending our techniques for supporting state and leveraging the fact that for an oblivious TM, the encryptor knows the time step at which a symbol will be next accessed. The second difference requires more care. It can be partly resolved by noticing that if computation ends, and no more SKE keys are output by $\mathsf{CktFE.Dec}$, then we may replace the SKE encryptions of $\mathsf{CktFE.CT}$ by encryptions of $\vec{0}$. These ciphertexts will be indistinguishable from the real world by semantic security of SKE.

The above argument holds for the SKE encryptions $\hat{\mathbf{D}}_i$ of the second CktFE ciphertext component $\mathbf{d}_i$, since it encodes state and CktFE decryption at step i only reveals SKE keys for step $i+1$. However, this does not apply to the case of $\hat{\mathbf{E}}_i$, i.e. the SKE encryptions of the tape symbol $\mathbf{e}_i$ since CktFE decryption at step i reveals SKE keys $\mathbf{K}'$ corresponding to step $t(i) > i$. Thus, it is possible that the decryptor receives SKE keys to decrypt some $\hat{\mathbf{E}}_{t(i)}$ where $t(i) > \mathsf{runtime}(M, \mathbf{w})$, but not does not receive SKE keys to decrypt $\hat{\mathbf{D}}_{t(i)}$. Thus, the decryptor can see an *incomplete part* of a CktFE ciphertext for time steps after $\mathsf{runtime}(M, \mathbf{w})$. We handle this issue by noticing that an incomplete ciphertext does not offer the adversary any advantage – formally, the CktFE simulator can be augmented to be invoked without an $M(\mathbf{w})$ value and it outputs incomplete ciphertexts. This is formalized in the following definition.

D.2.1 Augmented Security Definition for Circuit FE

We define $\mathsf{AUG\text{-}SIM}$ security, where a simulator may additionally be invoked without an $M(\mathbf{x})$ value to obtain a ciphertext component.

▶ **Definition 13** ($\mathsf{AUG\text{-}SIM}$- Security)**.** Let CktFE be a decomposable[5] functional encryption scheme for a circuit family $\mathcal{F}$. Say that a ciphertext $\mathsf{CT} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_n)$ as defined in

[5] This property is not required but allows for ease of exposition and suffices for our case.

Section C. For every p.p.t. adversary $A = (A_1, A_2)$ and a p.p.t. simulator Sim, and $k > 0$, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CktFE},A}(1^\kappa)}$:	$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CktFE},\mathrm{Sim}}(1^\kappa)}$:
1: $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{CktFE.Setup}(1^\kappa)$	1: $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{CktFE.Setup}(1^\kappa)$
2: $(f, st_1) \leftarrow A_1(\mathsf{PK})$	2: $(f, st_1) \leftarrow A_1(\mathsf{PK})$
3: $\mathsf{sk}_f \leftarrow \mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$	3: $\mathsf{sk}_f \leftarrow \mathsf{CktFE.Keygen}(\mathsf{MSK}, f)$
4: $(\mathbf{x}', st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_f)$	4: $(\mathbf{x}', st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_f)$
5: $\mathsf{CT}' \leftarrow \mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x}')$	5: $(\tilde{\mathsf{CT}}'_1, \ldots, \tilde{\mathsf{CT}}'_{n-k}) \leftarrow \mathrm{Sim}(\mathsf{PK}, \mathsf{sk}_f, f, 1^{\lvert\mathbf{x}'\rvert})$
6: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_f)$	6: $(\mathbf{x}, st) \leftarrow A_2(st_1, \mathsf{PK}, \mathsf{sk}_f)$
7: $\mathsf{CT} \leftarrow \mathsf{CktFE.Enc}(\mathsf{PK}, \mathbf{x})$	7: $\tilde{\mathsf{CT}} \leftarrow \mathrm{Sim}(\mathsf{PK}, \mathsf{sk}_f, f, f(\mathbf{x}), 1^{\lvert x\rvert})$
8: Output $(st, \mathsf{CT}, (\mathsf{CT}'_1, \ldots, \mathsf{CT}'_{n-k}))$	8: Output $(st, \mathsf{CT}, (\tilde{\mathsf{CT}}'_1, \ldots, \tilde{\mathsf{CT}}'_{n-k}))$

The functional encryption scheme CktFE is then said to be single query AUG-SIM secure if there exists a p.p.t. simulator Sim such that for every p.p.t. adversary $A = (A_1, A_2)$, the following two distributions are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CktFE},A}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CktFE},\mathrm{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

We note that the simulators of [18] and [27] can be modified to satisfy the above augmented notion of security. Intuitively, the simulator, given input $(\mathsf{PK}, \mathsf{sk}_f, f, f(\mathbf{x}), 1^{|x|})$ is now required to output a fraction of the ciphertext $\mathsf{CT}_\mathbf{x}$. For concreteness, let us assume it must output the last $n - k$ components of the ciphertext. Now, this partial ciphertext will never be decryptable by any function key, so the simulator may simply generate these components as $\tilde{\mathsf{CT}}'_i \leftarrow \mathcal{E}\ (\mathsf{PK}_i, 0, r, \hat{r}_i)$ for $i \in [k+1, n]$, where $\mathcal{E}$ is the encoding function used to generate a single ciphertext component, as defined in Section 3. Note that in the real world, we would correspondingly have $\mathsf{CT}'_i \leftarrow \mathcal{E}\ (\mathsf{PK}_i, x_i, r, \hat{r}_i)$.

In more detail, consider the ciphertext of [18]. As seen in Appendix C, $\mathcal{E}$ outputs an FHE ciphertext, an ABE ciphertext and a garbled circuit. To simulate a partial ciphertext, the simulator may generate $n - k$ FHE encryptions of 0, and use these as the attributes for the ABE scheme as described above. The garbled circuit and its labels are generated as before, the labels of the garbled circuit form the messages in the ABE ciphertext as before. Function keys are generated as before – recall these are ABE keys that select one of two labels embedded in each ABE CT depending on whether some predicate evaluates to true. Now consider an adversary who receives these $n - k$ components and the function key. He may decrypt the given ABE ciphertexts with the matching ABE predicate keys to obtain a subset of the labels of the garbled circuit. By semantic security of FHE and by security of garbled circuits, he cannot learn anything about the message $\mathbf{x}$ from these.

For [27], a ciphertext for input $\mathbf{x}$ corresponds to a fresh garbled circuit, PKE encryptions of both labels for the N wires corresponding to f and the n labels corresponding to $\mathbf{x}$ provided in the clear. A partial ciphertext in this case will correspond to revealing some subset of the n labels in the clear. The simulator can simply reveal labels corresponding to input 0 – by security of garbled circuits, the adversary cannot distinguish this view from the real world.

Formally, let us denote the incomplete ciphertext in either scheme as $\mathsf{CT}_{\mathsf{PART}}$. Then, given a distinguisher A who can distinguish between the real output $\big(\mathsf{PK}, C, \mathsf{SK}_C, \mathsf{CT}_{\mathsf{PART}}(\mathbf{x})\big)$ and the simulated output $\big(\mathsf{PK}, C, \mathsf{SK}_C, \mathsf{CT}_{\mathsf{PART}}(\mathbf{0})\big)$, we construct a distinguisher B who distinguishes between *complete* ciphertexts $\mathsf{CT}(\mathbf{x}_0)$ and $\mathsf{CT}(\mathbf{x}_1)$ where $C(\mathbf{x}_0) = C(\mathbf{x}_1)$, thus

violating FULL-SIM security of CktFE. B is constructed as follows: it chooses two messages $\mathbf{x}_0$ and $\mathbf{x}_1$ such that $C(\mathbf{x}_0) = C(\mathbf{x}_1)$ and they differ only in (fixed) $n-k$ positions – for these positions $\mathbf{x}_0$ may be arbitrary and $\mathbf{x}_1$ is $\mathbf{0}$. B submits these as "challenge" messages. He obtains PK, upon which he outputs C such that $C(\mathbf{x}_0) = C(\mathbf{x}_1)$. He receives SK_C as well as $\mathsf{CT}(\mathbf{x}_b)$ for a random bit b. B strips off the ciphertext components corresponding to the k positions which match and sends $\left(\mathsf{PK}, C, \mathsf{SK}_C, \mathsf{CT}_{\mathsf{PART}}(\mathbf{x}_b)\right)$ to A. B outputs whatever A outputs. It is clear that B has the same advantage as A.

D.2.2 Proof

▶ **Theorem 14.** *Assume that the underlying* CktFE *scheme satisfies augmented* FULL-SIM *security according to definition 13. Then the construction for* TMFE *achieves* FULL-SIM *security as defined in definition 10.*

Proof. We proceed to construct a simulator TMFE.Sim as required by Definition 10. When we invoke the augmented CktFE simulator to produce incomplete ciphertexts, we denote it by $\mathsf{CktFE.Sim_p}$. The simulator TMFE.Sim receives $(\mathsf{PK}, \mathsf{SK}_M, M, M(\mathbf{w}), 1^{|\mathbf{w}|}, \tau, \mathsf{runtime}(M, x))$ and does the following:

1. Assign the bit $b = M(\mathbf{w})$, let $s = \mathsf{runtime}(M, x)$, and construct the circuit f corresponding to M as defined in Figure 2 in the description of TMFE.KeyGen. Let $\mathsf{CktFE.SK_f} = \mathsf{SK_M}$.
2. Let $\mathsf{CktFE.Sim_p}$ be the Simulator defined in Definition 13 for $k = \log \Gamma$.
3. For $(i = \tau, i > s, i--)$, do:
   a. Choose $2 \log Q$ keys $\tilde{K}_{i,1}^{b_{ij}}, \ldots \tilde{K}_{i,\log Q}^{b_{ij}} \leftarrow \mathcal{K}^{2 \log Q}$ and compute
   $$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{b_{ij}} = \mathsf{SKE.Enc}(0^{|\tilde{\mathbf{d}}_{i,j}|}, \tilde{K}_{i,j}^{b_{ij}}) \ \ \forall\, j \in [\log Q], b_{ij} \in \{0,1\}$$
   b. Let $\mathsf{Sim}.\hat{\mathbf{D}}_{i,j} = \left(\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_{i,j}}, \ \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\bar{b}_{i,j}}\right)$ and $\mathsf{Sim}.\hat{\mathbf{D}}_i = \left(\mathsf{Sim}.\hat{\mathbf{D}}_{i,j} \right)$ for $j \in [\log Q]$.
   c. Let $(\tilde{\mathbf{c}}_i, \tilde{\mathbf{e}}_{i,\gamma_i}) = \mathsf{CktFE.Sim_p}(\mathsf{PK}, f, \mathsf{SK}_f)$
   d. Sample key $\mathbf{K}_i'^* = (K_{i,1}'^*, \ldots, K_{i,\log \Gamma}'^*) \leftarrow \mathcal{K}^{\log \Gamma}$ and let
   $$\mathsf{Sim}.\hat{\mathbf{e}}_{i,j} = \mathsf{SKE.Enc}(\tilde{\mathbf{e}}_{i,j}, K_{i,j}'^*) \ \ \forall j \in [\log \Gamma],$$
   e. Sample $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and assign $\mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\tilde{b}_{i,j}} = \mathsf{Sim}.\hat{\mathbf{e}}_{i,j}$
   f. Choose another $\log \Gamma$ keys $\tilde{K}'_{i,1}, \ldots \tilde{K}'_{i,\log \Gamma} \leftarrow \mathcal{K}^{\log \Gamma}$ and compute
   $$\mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\bar{b}_{ij}} = \mathsf{SKE.Enc}(0^{|\tilde{\mathbf{e}}_{i,j}|}, \tilde{K}'_{i,j}) \ \ \forall\, j \in [\log \Gamma]$$
   g. Let $\mathsf{Sim}.\hat{\mathbf{E}}_{i,j} = \left(\mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\tilde{b}_{i,j}}, \ \mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\bar{b}_{i,j}}\right)$ and $\mathsf{Sim}.\hat{\mathbf{E}}_i = \left(\mathsf{Sim}.\hat{\mathbf{E}}_{i,j} \right)$ for $j \in [\log \Gamma]$.
4. Invoke $\mathsf{CktFE.Sim}\left(\mathsf{PK}, f, \mathsf{CktFE.SK}_f, b \right)$ to receive $\tilde{\mathsf{CT}}_s$ where we may express $\tilde{\mathsf{CT}}_s = (\tilde{\mathbf{c}}_s, \tilde{\mathbf{d}}_{s,q_s}, \tilde{\mathbf{e}}_{s,\gamma_s})$, $\tilde{\mathbf{d}}_{s,q_s} = (\tilde{\mathbf{d}}_{s,j})$ for $j \in [\log Q]$ and $\tilde{\mathbf{e}}_{s,\gamma_s} = (\tilde{\mathbf{e}}_{s,j})$ for $j \in [\log \Gamma]$.
5. For $(i = s, i \geq 1, i--)$, do:
   a. If $i = 1$, set $\mathsf{Sim}.\hat{\mathbf{D}}_1 = \tilde{\mathbf{d}}_{1,q_1}$ and exit.
   b. Sample key $\mathbf{K}_i^* = (K_{i,1}^*, \ldots, K_{i,\log Q}^*) \leftarrow \mathcal{K}^{\log Q}$ and let
   $$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j} = \mathsf{SKE.Enc}(\tilde{\mathbf{d}}_{i,j}, K_{i,j}^*) \ \ \forall j \in [\log Q],$$
   c. Sample $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and assign $\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_{i,j}} = \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}$.

d. Choose another $\log Q$ keys $\tilde{K}_{i,1}, \dots \tilde{K}_{i,\log Q} \leftarrow \mathcal{K}^{\log Q}$ and compute

$$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\bar{b}_{ij}} = \mathsf{SKE.Enc}(0^{|\tilde{\mathbf{d}}_{i,j}|}, \tilde{K}_{i,j}) \;\; \forall\, j \in [\log Q]$$

e. Let $\mathsf{Sim}.\hat{\mathbf{D}}_{i,j} = \left(\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_{i,j}},\ \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\bar{b}_{i,j}}\right)$ and $\mathsf{Sim}.\hat{\mathbf{D}}_i = \left(\ \mathsf{Sim}.\hat{\mathbf{D}}_{i,j}\ \right)$ for $j \in [\log Q]$.
f. Sample key $\mathbf{K}_i'^* = (K_{i,1}'^*, \dots, K_{i,\log \Gamma}'^*) \leftarrow \mathcal{K}^{\log \Gamma}$ and let

$$\mathsf{Sim}.\hat{\mathbf{e}}_{i,j} = \mathsf{SKE.Enc}(\tilde{\mathbf{e}}_{i,j}, K_{i,j}'^*) \;\; \forall j \in [\log \Gamma],$$

g. Sample $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and assign $\mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\tilde{b}_{i,j}} = \mathsf{Sim}.\hat{\mathbf{e}}_{i,j}$
h. Choose another $\log \Gamma$ keys $\tilde{K}'_{i,1}, \dots \tilde{K}'_{i,\log \Gamma} \leftarrow \mathcal{K}^{\log \Gamma}$ and compute

$$\mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\bar{b}_{ij}} = \mathsf{SKE.Enc}(0^{|\tilde{\mathbf{e}}_{i,j}|}, \tilde{K}'_{i,j}) \;\; \forall\, j \in [\log \Gamma]$$

i. Let $\mathsf{Sim}.\hat{\mathbf{E}}_{i,j} = \left(\mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\tilde{b}_{i,j}},\ \mathsf{Sim}.\hat{\mathbf{e}}_{i,j}^{\bar{b}_{i,j}}\right)$ and $\mathsf{Sim}.\hat{\mathbf{E}}_i = \left(\ \mathsf{Sim}.\hat{\mathbf{E}}_{i,j}\ \right)$ for $j \in [\log \Gamma]$.
j. Let $(\tilde{\mathbf{c}}_{i-1}, \tilde{\mathbf{d}}_{i-1,q_{i-1}}, \tilde{\mathbf{e}}_{i-1,\gamma_{i-1}}) = \mathsf{CktFE.Sim}(\mathsf{PK}, f, \mathsf{SK}_f, \mathbf{K}_i^*, \mathbf{K}_{t(i-1)}'^*)$.

6. Output the simulated ciphertext as $\tilde{\mathsf{CT}}_{\mathbf{w}} = \Big(\big(\{\tilde{\mathbf{c}}_i, \mathsf{Sim}.\hat{\mathbf{D}}_i, \mathsf{Sim}.\hat{\mathbf{E}}_i\} \text{ for } i \in [\tau]\big),\ \tilde{\mathbf{d}}_1, \{\tilde{\mathbf{e}}_i\} \text{ for } i \in [1,k] \cup T\Big)$.

D.2.2.1 Analysis of Simulator.

Correctness of the simulator $\mathsf{TMFE.Sim}$ can be easily established using correctness of the simulator $\mathsf{CktFE.Sim_p}$ and the semantic security of SKE.

▶ **Theorem 15.** *Given that* $\mathbf{d}_{i,q_i} \overset{c}{\approx} \tilde{\mathbf{d}}_{i,q_i}$ *and* $\hat{\mathbf{D}}_i$ *and* $\mathsf{Sim}.\hat{\mathbf{D}}_i$ *are constructed according to* $\mathsf{TMFE.Enc}$ *and* $\mathsf{TMFE.Sim}$ *respectively, then* $\hat{\mathbf{D}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$.

The above theorem holds similarly for $\mathbf{e}_i$ and $\hat{\mathbf{E}}_i$ as well. The proof is as follows.

1. We now establish that $(\hat{\mathbf{d}}_{i,j}^{b_j} \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}_{i,j})$ where $b_j = q_{i,j}$ and $j \in [\log Q]$.

a. We have that in algorithm $\mathsf{TMFE.Enc}$,

$$\mathbf{K}_i = \Big((K^0_{(i,1)}, K^1_{(i,1)}), \dots, (K^0_{(i,\log Q)}, K^1_{(i,\log Q)})\Big)$$

where $K^b_{i,j} \leftarrow \mathcal{K}$ for $j \in [\log Q]$. We also have, for $j \in [\log Q]$, $b \in \{0,1\}$:

$$\hat{\mathbf{d}}^b_{i,j} = \mathsf{SKE.Enc}(K^b_{i,j}, \mathbf{d}^b_{i,j}) \tag{D.1}$$

b. In simulator $\mathsf{TMFE.Sim}$:

$$\mathbf{K}_i^* = (K^*_{i,1}, \dots, K^*_{i,\log Q}) \leftarrow \mathcal{K}^{\log Q} \quad \text{and}$$

$$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j} = \mathsf{SKE.Enc}(K^*_{i,j}, \tilde{\mathbf{d}}_{i,j}) \;\; \forall j \in [\log Q],$$

Hence, since $\mathbf{d}^{b_j}_{i,j} \overset{c}{\approx} \tilde{\mathbf{d}}_{i,j}$ and the symmetric keys are picked using the same distribution in each case, we have that $(\hat{\mathbf{d}}^{b_j}_{i,j} \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}_{i,j})$ where $b_j = q_{i,j}$ and $j \in [\log Q]$.

2. We now establish that $(\hat{\mathbf{d}}^{\bar{b}_j}_{i,j} \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}^{\bar{b}_j}_{i,j})$ where $j \in [\log Q]$.

a. Construction of $\hat{\mathbf{d}}^{\bar{b}_j}_{i,j}$ is described in Equation D.1.

b. For the latter, TMFE.Sim samples $\tilde{b}_j$ and sets $\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_j} = \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}$. Next, it samples another $\log Q$ keys $\tilde{K}_{i,1}, \ldots \tilde{K}_{i,\log Q} \leftarrow \mathcal{K}^{\log Q}$ and computes

$$\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\bar{\tilde{b}}_j} = \mathsf{SKE.Enc}(\tilde{K}_{i,j}, 0^{|\tilde{\mathbf{d}}_{i,j}|}) \ \ \forall\, j \in [\log Q]$$

By semantic security of SKE, we have that $(\hat{\mathbf{d}}_{i,j}^{\bar{b}_j} \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\bar{\tilde{b}}_j})$.

3. Next, we show that $\hat{\mathbf{D}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$. For $i = 1$, we have by definitions of $\hat{\mathbf{D}}_1$ and $\mathsf{Sim}.\hat{\mathbf{D}}_1$, that the above holds. For $i > 1$, in TMFE.Enc, we have $b_{i,j} \leftarrow \{0,1\}$ and

$$\hat{\mathbf{D}}_{i,j} = (\ \hat{\mathbf{d}}_{i,j}^{b_{i,j}}, \hat{\mathbf{d}}_{i,j}^{\bar{b}_{i,j}}\ )$$

In TMFE.Sim, we have $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and

$$\mathsf{Sim}.\hat{\mathbf{D}}_{i,j} = \big(\mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_{i,j}},\ \mathsf{Sim}.\hat{\mathbf{d}}_{i,j}^{\bar{\tilde{b}}_{i,j}}\big)$$

Since $\hat{\mathbf{D}}_i = (\hat{\mathbf{D}}_{i,j})$ and $\mathsf{Sim}.\hat{\mathbf{D}}_i = (\ \mathsf{Sim}.\hat{\mathbf{D}}_{i,j}\ )$ for $j \in [\log Q]$, we have that $\hat{\mathbf{D}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$.

Let us say that the TM M visits states $q_1, \ldots, q_s$ while computing on input $\mathbf{w}$ where $|\mathbf{w}| = k$.

1. For $s < i \leq \tau$

a. We have by correctness of $\mathsf{CktFE.Sim_p}$, that:

$$\Big\{(\mathbf{c}_i, \mathbf{e}_{i,\gamma_i}) \leftarrow \mathsf{CktFE.Enc}\big(\mathsf{PK}, (\mathbf{K}_{i+1}, \mathbf{K}'_{t(i)}, q_i, \gamma_i)\big)$$
$$\overset{c}{\approx} (\tilde{\mathbf{c}}_i, \tilde{\mathbf{e}}_{i,\gamma_i}) \leftarrow \mathsf{CktFE.Sim_p}\big(\mathsf{PK}, f_M, \mathsf{SK}_f\big)\Big\}$$

b. By Theorem 15 we get that $\hat{\mathbf{E}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{E}}_i$ and by semantic security of SKE, we get that $\hat{\mathbf{D}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$. Hence, $(\mathbf{c}_i, \hat{\mathbf{D}}_i, \hat{\mathbf{E}}_i) \overset{c}{\approx} (\tilde{\mathbf{c}}_i, \mathsf{Sim}.\hat{\mathbf{D}}_i, \mathsf{Sim}.\hat{\mathbf{E}}_i)$.

2. We have by correctness of CktFE.Sim according to definition 7 that:

$$\Big\{(\mathbf{c}_s, \mathbf{d}_{s,q_s}, \mathbf{e}_{s,\gamma_s}) \leftarrow \mathsf{CktFE.Enc}\big(\mathsf{PK}, (\mathbf{K}_{s+1}, \mathbf{K}'_{t(s)}, q_s, \gamma_s)\big)$$
$$\overset{c}{\approx} (\tilde{\mathbf{c}}_s, \tilde{\mathbf{d}}_{s,q_s}, \tilde{\mathbf{e}}_{s,\gamma_s}) \leftarrow \mathsf{CktFE.Sim}\big(\mathsf{PK}, f_M, \mathsf{SK}_f, b\big)\Big\}$$

By decomposability, $\mathsf{CT}'_s = (\mathbf{c}_s, \mathbf{d}_{s,q_s}, \mathbf{d}_{s,q_s})$ where $\mathbf{d}_{s,q_s} = (\mathbf{d}_{s,j}^{b_j})$ for $j \in [\log Q]$ and $b_j = q_{s,j}$, and similarly $\mathbf{e}_{s,\gamma_s} = (\mathbf{e}_{s,j}^{b_j})$ for $j \in [\log \Gamma]$ and $b_j = \gamma_{s,j}$ Similarly, the last two terms of $\tilde{\mathsf{CT}}'_s = (\tilde{\mathbf{c}}_s, \tilde{\mathbf{d}}_{s,q_s}, \tilde{\mathbf{e}}_{s,\gamma_s})$ can be decomposed into $(\tilde{\mathbf{d}}_{s,j})$ and $(\tilde{\mathbf{e}}_{s,j})$. Let i = s. Theorem 15 gives us that $\hat{\mathbf{D}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{D}}_i$ and $\hat{\mathbf{E}}_i \overset{c}{\approx} \mathsf{Sim}.\hat{\mathbf{E}}_i$ and hence $(\mathbf{c}_i, \hat{\mathbf{D}}_i, \hat{\mathbf{E}}_i) \overset{c}{\approx} (\tilde{\mathbf{c}}_i, \mathsf{Sim}.\hat{\mathbf{D}}_i, \mathsf{Sim}.\hat{\mathbf{E}}_i)$.

3. Let $i = i - 1$. Now, we have by correctness of CktFE.Sim according to Definition 7,

$$\Big\{(\mathbf{c}_i, \mathbf{d}_{i,q_i}, \mathbf{e}_{i,\gamma_i}) \leftarrow \mathsf{CktFE.Enc}\big(\mathsf{PK}, (\mathbf{K}_{i+1}, \mathbf{K}'_{t(i)}, q_i, \gamma_i)\big)$$
$$\overset{c}{\approx} (\tilde{\mathbf{c}}_i, \tilde{\mathbf{d}}_{i,q_i}, \tilde{\mathbf{e}}_{i,\gamma_i}) \leftarrow \mathsf{CktFE.Sim}\big(\mathsf{PK}, f_M, \mathsf{SK}_f, (\mathbf{K}^*_{i+1}, \mathbf{K}'^*_{t(i)})\big)\Big\}$$

$\mathsf{CT}'_i = (\mathbf{c}_i, \mathbf{d}_{i,q_i}, \mathbf{d}_{i,q_i})$ where $\mathbf{d}_{i,q_i} = (\mathbf{d}_{i,j}^{b_j})$ for $j \in [\log Q]$ and $b_j = q_{i,j}$, and similarly $\mathbf{e}_{i,\gamma_i} = (\mathbf{e}_{i,j}^{b_j})$ for $j \in [\log \Gamma]$ and $b_j = \gamma_{i,j}$ Similarly, the last two terms of $\tilde{\mathsf{CT}}'_i = (\tilde{\mathbf{c}}_i, \tilde{\mathbf{d}}_{i,q_i}, \tilde{\mathbf{e}}_{i,\gamma_i})$ can be decomposed into $(\tilde{\mathbf{d}}_{i,j})$ and $(\tilde{\mathbf{e}}_{i,j})$.

4. Hence, from Theorem 15, $(\mathbf{c}_i, \hat{\mathbf{D}}_i, \hat{\mathbf{E}}_i) \overset{c}{\approx} (\tilde{\mathbf{c}}_i, \mathsf{Sim}.\hat{\mathbf{D}}_i, \mathsf{Sim}.\hat{\mathbf{E}}_i)$.

5. If $i \geq 1$, go to step 3.
6. Now, a straightforward hybrid argument yield that:

$$\left\{(\mathbf{c}_1, \hat{\mathbf{D}}_1, \hat{\mathbf{E}}_1), (\mathbf{c}_2, \hat{\mathbf{D}}_2, \hat{\mathbf{E}}_2), \ldots, (\mathbf{c}_\tau, \hat{\mathbf{D}}_\tau, \hat{\mathbf{D}}_\tau)\right\}$$
$$\overset{c}{\approx} \left\{(\tilde{\mathbf{c}}_1, \mathsf{Sim}.\hat{\mathbf{D}}_1, \mathsf{Sim}.\hat{\mathbf{E}}_1), (\tilde{\mathbf{c}}_2, \mathsf{Sim}.\hat{\mathbf{D}}_2, \mathsf{Sim}.\hat{\mathbf{E}}_2), \ldots, (\tilde{\mathbf{c}}_\tau, \mathsf{Sim}.\hat{\mathbf{D}}_\tau, \mathsf{Sim}.\hat{\mathbf{E}}_\tau)\right\}$$

Further using the fact that $\mathbf{d}_1 \overset{c}{\approx} \tilde{\mathbf{d}}_1$ and $\mathbf{e}_i \overset{c}{\approx} \tilde{\mathbf{e}}_i$, we prove $\left((\{\mathbf{c}_i, \hat{\mathbf{D}}_i, \hat{\mathbf{E}}_i\} \text{ for } i \in [\tau]), \mathbf{d}_1, \{\mathbf{e}_i\} \text{ for } i \in [1,k] \cup T\right) \overset{c}{\approx} \left((\{\tilde{\mathbf{c}}_i, \mathsf{Sim}.\hat{\mathbf{D}}_i, \mathsf{Sim}.\hat{\mathbf{E}}_i\} \text{ for } i \in [\tau]), \tilde{\mathbf{d}}_1, \{\tilde{\mathbf{e}}_i\} \text{ for } i \in [1,k] \cup T\right)$ as desired.

◀

E Reusable Garbled DFAs

In this section, we will define garbled DFAs and notions of input and function privacy, adapting corresponding definitions from [18]. We further show how to construct garbled DFAs (with unbounded inputs) that can be used to evaluate multiple inputs (of possibly varying size).

▶ **Definition 16.** (Garbled DFA scheme) A garbling scheme for a family of DFAs $\mathcal{M} = \{\mathcal{M}_\kappa\}_{\kappa \in \mathbb{N}}$ with M_κ a family of DFAs $\Sigma_\kappa \times \mathcal{Q}_\kappa \to \mathcal{Q}_\kappa$, is a tuple of p.p.t. algorithms $\mathsf{GbDfa} = (\mathsf{GbDfa.Garble}, \mathsf{GbDfa.Enc}, \mathsf{GbDfa.Eval})$ such that

- $\mathsf{GbDfa.Garble}(1^\kappa, \mathsf{M})$ takes as input the security parameter κ and a DFA $M \in \mathcal{M}_\kappa$ and outputs the garbled DFA M_G and a secret key gsk.
- $\mathsf{GbDfa.Enc}(\mathsf{gsk}, \mathbf{w})$ takes as input the vector $\mathbf{w} \in \Sigma^*$, the secret key gsk and outputs an encoding c.
- $\mathsf{GbDfa.Eval}(\mathsf{M_G}, \mathsf{c})$ takes as input a garbled DFA M_G, an encoding c and outputs 1 iff M accepts $\mathbf{w}$, 0 otherwise.

▶ **Definition 17.** (Correctness). For all sufficiently large security parameters κ, for all DFAs $M \in \mathcal{M}_\kappa$ and all $\mathbf{w} \in \Sigma^*$, we have:

$$\Pr\left[\begin{array}{l}(M_G, \mathsf{gsk}) \leftarrow \mathsf{GbDfa.Garble}(1^\kappa, M);\\ c \leftarrow \mathsf{GbDfa.Enc}(\mathsf{gsk}, \mathbf{w}); b \leftarrow \mathsf{GbDfa.Eval} : M(\mathbf{w}) = b\end{array}\right] = 1 - \mathrm{negl}(\kappa)$$

▶ **Definition 18.** (Efficiency) There exist universal polynomials $p_1 = p_1(\kappa)$ and $p_2 = p_2(\kappa, \cdot)$ such that for all security parameters κ, for all DFAs $M \in M_\kappa$, for all $\mathbf{w} \in \Sigma^*$,

$$\Pr\left[\begin{array}{l}(M_G, \mathsf{gsk}) \leftarrow \mathsf{GbDfa.Garble}(1^\kappa, M) :\\ |\mathsf{gsk}| \leq p_1(\kappa) \text{ and runtime } (\mathsf{GbDfa.Enc}(\mathsf{gsk}, \mathbf{w})) \leq p_2(\kappa, |\mathbf{w}|)\end{array}\right] = 1.$$

▶ **Definition 19.** (Input and machine privacy with reusability) Let GbDfa be a garbling scheme for a family of DFAs $\mathcal{M} = \{\mathcal{M}_\kappa\}_{\kappa \in \mathbb{N}}$. For a pair of p.p.t. algorithms $A = (A_1, A_2)$ and a p.p.t. simulator $S = (S_1, S_2)$, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{GbDfa},\mathsf{A}}(1^\kappa)\text{:}}$

1: $(M, st_A) \leftarrow A_1(1^\kappa)$
2: $(\mathsf{gsk}, M_G) \leftarrow \mathsf{GbDfa.Garble}(1^\kappa, \mathsf{M})$
3: $\alpha \leftarrow A_2^{\mathsf{GbDfa.Enc}(\mathsf{gsk},\cdot)}(M, M_G, st_A)$
4: Output α

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{GbDfa},\mathsf{A},\mathsf{S}}(1^\kappa)\text{:}}$

1: $(M, st_A) \leftarrow A_1(1^\kappa)$
2: $(\tilde{M}_G, st_S) \leftarrow S_1(1^\kappa, 1^{|M|})$
3: $\alpha \leftarrow A_2^{\mathsf{O}(\cdot,M)[[st_S]]}(M, \tilde{M}_G, st_A)$
4: Output α

Here, $\mathsf{O}(\cdot, M)[[st_s]]$ is an oracle that on input $\mathbf{w}$ from A_2, runs S_2 with inputs $M(\mathbf{w}), 1^{|\mathbf{w}|}$, and the latest state of S; it returns the output of S_2 (storing the new simulator state for the next invocation).

A garbling scheme GbDfa is input and machine private with reusability if there exists a p.p.t. simulator S such that for all pairs of p.p.t. adversaries $A = (A_1, A_2)$, the following two distributions are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{GbDfa},\mathsf{A}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{GbDfa},\mathsf{A},\mathsf{S}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

E.1 Construction

Our DfaFE scheme provides input privacy and reusability but not machine privacy. To achieve a reusable, garbled DFA, we need to hide the DFA machine as well. [18] combats this problem for circuits by using a semantically secure symmetric encryption scheme to encrypt the circuit C. The resultant encryption, say $\hat{C}$, is embedded into a universal circuit U_E that takes as input a pair $(\mathbf{x}, \mathsf{SK_S})$, where $\mathbf{x}$ is the original input to C, and $\mathsf{SK_S}$ the symmetric secret key. It then proceeds to decrypt $\hat{C}$ using $\mathsf{SK_S}$ and computes $C(\mathbf{x})$. The underlying FE scheme for circuits is used to encrypt the pair $(\mathbf{x}, \mathsf{SK_S})$ and the functional key is generated for the circuit U_E.

In the case of functional encryption for DFAs however, this approach does not work directly. To see this, suppose we encrypt the machine M using symmetric key encryption. Then it is unclear how to construct a DFA that runs the symmetric key decryption circuit.

However, the blueprint from [18] does work for our specific construction. This is because our DFA functional key is a circuit functional key for an appropriately defined circuit (refer Figure 1). Thus, our particular DFA functional key is indeed amenable to the hiding technique of [18]. We proceed to provide the detailed construction.

We use a semantically secure symmetric key encryption scheme $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$. The reusable garbled DFA scheme is defined as follows.

$\mathsf{GbDfa.Garble}(1^\kappa, \mathsf{M})$: Upon input the security parameter and a machine M, do the following:

1. Choose a symmetric key encryption scheme SKE with key space $\mathcal{K}$.
2. Generate a secret key $\mathsf{SK_S} \leftarrow \mathsf{SKE.KeyGen}(1^\kappa)$.
3. Define a circuit family as follows. Let $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} = (\mathcal{K} \times \Sigma \times \mathcal{K}^{2\log Q} \times \{0,1\}) \times \mathcal{Q}$ and $\mathcal{Y} = \mathcal{K}^{\log Q}$. We set
$$\ell_1 = \lfloor\mathcal{K}\rfloor + \lfloor\Sigma\rfloor + \lfloor\mathcal{K}^{2\log Q}\rfloor + 1, \quad \ell_2 = \lfloor\mathcal{Q}\rfloor$$
where $\lfloor\cdot\rfloor$ denotes size in bits. Let $\ell = \ell_1 + \ell_2$.
4. Invoke $\mathsf{CktFE.Setup}(1^\kappa, 1^\ell)$ to obtain $\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_{2,1}, \ldots, \mathsf{PK}_{2,\log Q}, \mathsf{PK}_{\mathsf{indpt}})$ and MSK.
5. Let f_M be the circuit defined in Figure 1 corresponding to machine M.
6. Let $E = \mathsf{SKE.Enc}(\mathsf{SK_S}, f_M)$. Instead of running $\mathsf{CktFE.Keygen}$ on f_M, we run it on a universal circuit U_E defined in Figure 3.
7. Let $M_G = \mathsf{CktFE.KeyGen}(\mathsf{U_E}, \mathsf{MSK})$ be the garbled DFA and $\mathsf{gsk} = (\mathsf{PK}, \mathsf{SK_S})$ be the secret key.

Universal Circuit U_E

U_E takes as input a secret key $\mathsf{SK_S}$ and a string $\mathbf{w}$:

- Compute $f_M = \mathsf{SKE.Dec}(\mathsf{SK_S}, E)$.
- Run f_M on $\mathbf{w}$.

Figure 3 Universal Circuit.

$\mathsf{GbDfa.Enc}(\mathsf{gsk}, \mathbf{w})$: Upon input a secret key gsk and a vector $\mathbf{w}$, do the following: Parse gsk as $(\mathsf{PK}, \mathsf{SK_S})$. Let $|\mathbf{w}| = k$. Output $c = \mathsf{DfaFE.Enc}(\mathsf{PK}, \mathbf{w}')$ where $\mathbf{w}' = \big((\mathsf{SK_S}, w_1), \ldots, (\mathsf{SK_S}, w_k)\big)$

$\mathsf{GbDfa.Eval}(\mathsf{M_G}, \mathsf{c})$: Compute and output $\mathsf{DfaFE.Dec}(M_G, c)$.

E.2 Correctness and Efficiency

Correctness is immediate from correctness of DfaFE and CktFE. To see this, note that the only changes in the above construction from the construction in Section 4 are:

1. Replacing the circuit f_M defined in figure 1 to U_E defined in figure 3 in the description of DfaFE.KeyGen.
2. Replacing the message $\mathbf{w} = (w_1, \ldots, w_k)$ with $\mathbf{w}' = \big((\mathsf{SK_S}, w_1'), \ldots, (\mathsf{SK_S}, w_k')\big)$ in the description of DfaFE.enc.

Thus, it suffices to establish that the pair $(U_E, \mathbf{w}')$ produces exactly the same value as the pair $(f_M, \mathbf{w})$. The remainder of correctness follows directly from Section 4.1. Let

$$\forall i \in [k-1],\ \mathsf{CktFE.Enc}\Big(\ \mathsf{PK}, \big((\mathsf{SK_S}, w_i), \mathbf{K}_{i+1}, 0, q_i\big)\ \Big) = (\mathbf{c}_i, \mathbf{d}_i)$$

$$\begin{aligned}
\mathsf{CktFE.Dec}(M_G, (\mathbf{c}_i, \mathbf{d}_i)) &= U_E(\mathsf{SK_S}, w_i, \mathbf{K}_{i+1}, 0, q_i)\ \big)\ \text{by correctness of CktFE} \\
&= C(w_i, \mathbf{K}_{i+1}, 0, q_i)\ \big)\ \text{by definition of } U_E \\
&= f_M(w_i, \mathbf{K}_{i+1}, 0, q_i)\ \text{ for }\ i < k, \\
&= f_M(w_k, \bot, 1, q_k)\ \text{ for }\ i = k.
\end{aligned}$$

Thus we have that at every step the DFA computation results in the same value regardless of whether the pair $(U_E, \mathbf{w}')$ or $(f_M, \mathbf{w})$ is used. Since the remainder of the computation proceeds as the construction in Section 4, we have the correctness of GbDfa.Eval from the correctness of DfaFE.Dec.

We now establish efficiency. We have that $|\mathsf{gsk}|$ equals $|\mathsf{SK_S}| + |\mathsf{DfaFE.PK}|$ where the DfaFE public key is a CktFE public key for circuits of input length $|\mathsf{SK_S}| + \log(|\mathcal{K}^{2\log Q}|) + \log(|\Sigma|) + \log(|\mathcal{Q}|)$. The length of the CktFE public key is polynomial in the input length of the circuit. Hence $|\mathsf{gsk}| = O(\mathrm{poly}(\kappa))$.

Next, consider the running time of the GbDfa.Enc algorithm. This algorithm, upon input of size k, invokes the CktFE.Enc algorithm k times, with polynomial sized messages. Hence, its runtime is $O(\kappa, k)$.

E.3 Security

We now establish that the scheme GbDfa constructed is secure according to Definition 19.

▶ **Theorem 20.** *Assume that the* DfaFE *construction described in Section 4 is* FULL-SIM *secure according to definition 2. Then the reusable garbled DFA construction provided in Section E.1 achieves input and machine privacy with reusability as defined in definition 19.*

Proof. We proceed to construct a simulator to satisfy the definition 19.

E.3.1 Simulator Construction.

We construct a simulator $S = (S_1, S_2)$ as follows.

$S_1(1^\kappa, 1^{|M|})$: Upon input the security parameter and the machine size, S_1 does:

1. Generate $(\mathsf{SKE.SK_S}, \mathsf{DfaFE.MSK})$ and $\mathsf{DfaFE.PK}$ as in $\mathsf{GbDfa.Garble}$.
2. Compute $\tilde{E} = \mathsf{SKE.Enc}(\mathsf{SK_S}, 0^{|M|})$
3. Output $\tilde{M}_G = \mathsf{CktFE.Enc}(U_{\tilde{E}}, \mathsf{MSK})$, where $U_{\tilde{E}}$ is as defined in Figure 3 but for $\tilde{E}$.
4. Set state $st_S = (\mathsf{SK_S}, U_{\tilde{E}}, \mathsf{DfaFE.PK}, \tilde{M}_G)$. Note that $\mathsf{DfaFE.PK} = \mathsf{CktFE.PK}$ for the family of circuits as defined in Section 4.

The simulator S_2 receives $(st_S, M(\mathbf{w}), 1^{|\mathbf{w}|})$ and does the following:

1. Assign the bit $b = M(\mathbf{w})$.
2. Let $\mathsf{CktFE.SK_{U_{\tilde{E}}}} = \tilde{\mathsf{M}}_\mathsf{G}$ and invoke $\mathsf{CktFE.Sim}\big(\mathsf{CktFE.PK}, U_{\tilde{E}}, \mathsf{CktFE.SK}_{U_{\tilde{E}}}, b\big)$ to receive $\tilde{\mathsf{CT}}_k$ where we may express $\tilde{\mathsf{CT}}_k = (\tilde{\mathbf{c}}_k, \tilde{\mathbf{d}}_{k,q_k})$ and $\tilde{\mathbf{d}}_{k,q_k} = (\tilde{\mathbf{d}}_{k,j})$ for $j \in [\log Q]$.
3. For $(i = k, i \geq 1, i--)$, do:
   a. If $i = 1$, set $S_2.\hat{\mathbf{D}}_1 = \tilde{\mathbf{d}}_{1,q_1}$ and exit.
   b. Sample key $\mathbf{K}_i^* = (K_{i,1}^*, \ldots, K_{i,\log Q}^*) \leftarrow \mathcal{K}^{\log Q}$ and let
   $$S_2.\hat{\mathbf{d}}_{i,j} = \mathsf{SKE.Enc}(\tilde{\mathbf{d}}_{i,j}, K_{i,j}^*) \;\; \forall j \in [\log Q],$$
   c. Sample $\tilde{b}_{i,j} \leftarrow \{0,1\}$ and assign $S_2.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_{i,j}} = S_2.\hat{\mathbf{d}}_{i,j}$.
   d. Choose another $\log Q$ keys $\tilde{K}_{i,1}, \ldots \tilde{K}_{i,\log Q} \leftarrow \mathcal{K}^{\log Q}$ and compute
   $$S_2.\hat{\mathbf{d}}_{i,j}^{\bar{b}_{ij}} = \mathsf{SKE.Enc}(0^{|\tilde{\mathbf{d}}_{i,j}|}, \tilde{K}_{i,j}) \;\; \forall\, j \in [\log Q]$$
   e. Let $S_2.\hat{\mathbf{D}}_{i,j} = \big(S_2.\hat{\mathbf{d}}_{i,j}^{\tilde{b}_{i,j}},\ S_2.\hat{\mathbf{d}}_{i,j}^{\bar{b}_{i,j}}\big)$ and $S_2.\hat{\mathbf{D}}_i = \big(\ S_2.\hat{\mathbf{D}}_{i,j}\ \big)$ for $j \in [\log Q]$.
   f. Let $(\tilde{\mathbf{c}}_{i-1}, \tilde{\mathbf{d}}_{i-1,q_{i-1}}) = \mathsf{CktFE.Sim}\big(\mathsf{CktFE.PK}, U_{\tilde{E}}, \mathsf{CktFE.SK}_{U_{\tilde{E}}}, \mathbf{K}_i^*\big)$.
4. Output the encoding as $c(\mathbf{w}) = (\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2, \ldots, \tilde{\mathbf{c}}_k, S_2.\hat{\mathbf{D}}_1, \ldots, S_2.\hat{\mathbf{D}}_k)$.

E.3.2 Analysis of Simulator.

We now establish that the simulator described above is correct.

We need to show that:

$$\left\{(M, st_A, M_G, \{\mathbf{w}_i, c(\mathbf{w}_i)\}_{i=0}^t) \leftarrow \mathsf{Exp}^{\mathsf{real}}_{\mathsf{GbDfa,A}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx}$$
$$\left\{(M, st_A, \tilde{M}_G, \{\mathbf{w}_i, \tilde{c}(\mathbf{w}_i)\}_{i=0}^t) \leftarrow \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{GbDfa,A,S}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

The proof is analogous to [18, Sec 4]. Indistinguishability is established via a sequence of games where the first game is the ideal distribution and the final game is the real

distribution. We show that successive games are indistinguishable, which implies that the above distributions are indistinguishable.

Hybrid 0: The ideal hybrid of Definition 19 with simulator S. In this case, the output distribution is:

$$\left(M, st_A, \tilde{M}_G, \{\mathbf{w}_i, S_2(st_S, M(\mathbf{w}), 1^{|\mathbf{w}|})\}_{i=0}^{t} \right)$$

Hybrid 1: The same as Hybrid 0, but $\tilde{E}$ is replaced with $E = \mathsf{SKE.Enc}(\mathsf{SK_S}, C)$. The output distribution is

$$\left(M, st_A, M_G, \{\mathbf{w}_i, S_2(st_S, M(\mathbf{w}), 1^{|\mathbf{w}|}\ )\}_{i=0}^{t} \right)$$

Hybrid 2: The real distribution using the GbDfa construction. The output distribution is

$$\left(M, st_A, M_G, \{\mathbf{w}_i, c(\mathbf{w}_i)\}_{i=0}^{t} \right)$$

Hybrid 0 and Hybrid 1 are computationally indistinguishable because of the semantic security of SKE.Enc. We will prove that Hybrid 1 and Hybrid 2 are computationally indistinguishable by using the FULL-SIM security of DfaFE.

Assume we have a p.p.t. adversary $A = (A_1, A_2)$ and p.p.t. distinguisher D that can distinguish between Hybrid 1 and Hybrid 2. We construct $A^{\mathsf{FE}} = (A_1^{\mathsf{FE}}, A_2^{\mathsf{FE}})$ and D^{FE} which breaks a modified version of the security of DfaFE in Definition 7. We will let an adversary query for multiple ciphertexts (i.e. repeat steps 4-5 as many times as it wants) and also give the simulator the circuit corresponding to M instead of the machine (in step 5 instead of M, a circuit as defined in Figure 1 is given to the simulator). Under this definition S_2 is a valid simulator for DfaFE (it's proof of security will be exactly as for DfaFE.Sim). An adversary that can break this definition can break the original definition with a polynomially smaller advantage.

On input PK, adversary A^{FE} will do the following:

1. Run A_1 on input 1^κ and get M and the corresponding circuit of function f defined in Figure 1 from st_A.
2. Generate $\mathsf{SK_S} \leftarrow \mathsf{SKE.KeyGen}(1^\kappa)$ and encrypt $E \leftarrow \mathsf{SKE.Enc}(\mathsf{SK_S}, M)$.
3. Let U_E be the circuit defined in Figure 3.
4. Output function U_E and $st_A^{\mathsf{FE}} = (\mathsf{SK_S}, U_E, st_A)$

On input $(\mathsf{SK}_{U_E}, st_A^{\mathsf{FE}})$, adversary A_2^{FE} does:

1. Set $M_G = \mathsf{SK}_{U_E}$
2. Run A_2 with input (U_E, M_G, st_A) by responding to its ith oracle query $\mathbf{w}_i$ with either of the following:
   - Return the the real ciphertext $\mathsf{CT}(\mathbf{w}_i) = \mathsf{DfaFE.Enc}(\mathsf{PK}, \mathbf{w}_i')$ where $\mathbf{w}_i' = \big((\mathsf{SK_S}, \mathbf{w}_{i,1}), \ldots, (\mathsf{SK_S}, \mathbf{w}_{i,|\mathbf{w}_i|})\big)$
   - Return the simulated ciphertext $\tilde{\mathsf{CT}}(\mathbf{w}_i) \leftarrow S_2(st_S, M(\mathbf{w}), 1^{|\mathbf{w}|})$ where $st_S = (\mathsf{PK}, M_G, U_E)$.

   Repeat this (sending either all real or all simulated ciphertexts) till A_2 outputs α.
3. Output α

When real ciphertexts are used then A^{FE} simulates Hybrid 2 and when simulated cipthertexts are used then they simulate Hybrid 1. Hence, a distinguisher that can distinguish between Hybrid 1 and 2 can also distinguish between $\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{DfaFE,A}}(1^\kappa)\}_{\kappa\in\mathbb{N}}$ and $\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{DfaFE,S_2}}(1^\kappa)\}_{\kappa\in\mathbb{N}}$ which contradicts FULL-SIM security of DfaFE.

◀