FINE-GRAINED ENCRYPTION FOR UNIFORM MODELS OF COMPUTATION

A THESIS

submitted by

MONOSIJ MAITRA

for the award of the degree

of

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MARCH 2020

THESIS CERTIFICATE

This is to certify that the thesis titled **FINE-GRAINED ENCRYPTION FOR UNIFORM MODELS OF COMPUTATION**, submitted by **Monosij Maitra**, to the Indian Institute of Technology, Madras, for the award of the degree of **Doctor of Philosophy**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Shweta Agrawal
Research Guide
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology Madras, 600036.

Place: Chennai

Date: 26th March, 2020

To all who matter.

ACKNOWLEDGEMENTS

My PhD journey has been an interesting one right from the start. I got admission in IIT Madras with a curiosity to learn about and do research in cryptography, about which I had very little idea at first. I am extremely thankful to Prof. C. Pandu Rangan that he still agreed to supervise me (with great enthusiasm) jointly with Dr. Rajsekar Manokaran, who was my principal advisor initially. It took me some time to understand the basics of the subject through some formal courses and some amount of self-reading. Slightly more than a year later, I decided to change my supervisor when Rajsekar left and Dr. Shweta Agrawal joined IIT Madras around the same time. The research phase in my PhD began mainly with Shweta being my supervisor. There have been loads of fun times as well as that of struggle and anxiety too in between but all of them proved to be essential ingredients to help me evolve in life. All those times moulded me immensely in changing my perspectives and also to expand my perceptions about many things, not necessarily restricted only to academics. The last four and a half years have been the finest learning and fun phase of my life till now and I would not want to barter it with anything else.

First and foremost, I consider myself to be exceptionally fortunate to have Shweta as my supervisor. This thesis would have been incomplete without her active involvement. She has nurtured me from the beginning of my research in PhD similar to how a mother takes care of her child. In my experience till now, I have always sensed a perpetual compassion and an intense involvement from her towards everything and everybody, whoever comes in touch with her. Apart from her integrity, boldness, outspoken and energetic nature, what I admire most is her clarity of thinking. Going ahead in life, I would consider myself quite lucky if I can build it in myself even to some small extent and help others to do so. She has been providing me rare insights into many aspects of life, aside from academic ones, for which I do not have enough words to convey my gratitude. My experience as her student will always remain as a treasure trove for me.

I always had an interest in teaching even before coming to IIT Madras. This was further stimulated by many professors in the Department of Computer Science and

Engineering (CSE), IIT Madras in various courses that I took as well as the ones for which I served as a teaching assistant. Of special mention is Prof. C. Pandu Rangan's teaching. As a student, I consider myself lucky to have sat through and experienced his lectures. My sincerest gratitude to him for his inspiring and absorbing way of teaching.

I am thankful to all the staffs and faculties in CSE for the help I got from them during various situations. In particular, I would like to thank my doctoral committee (DC) members: Dr. Meghana Nasre, Dr. Jayalal Sarma, Prof. P. Sreenivasa Kumar (chairperson) from CSE as well as Prof. Andrew Thangaraj from the Department of Electrical Engineering (EE) for their support, insightful comments and questions during my seminars and DC meetings.

I gratefully acknowledge Dr. Nishanth Chandran from Microsoft Research India, Prof. Damien Stehlé and Prof. Benoît Libert from ENS de Lyon for offering me an internship in their respective research groups and providing me with generous financial support during these periods. I thank them all for enhancing my experience in research. I would also like to sincerely acknowledge both my co-authors Dr. Shota Yamada from AIST Japan as well as Shweta for enriching me immensely about writing research papers.

I would like to thank all my lab-mates from Theoretical Computer Science (TCS) lab for spending long hours on various stimulating discussions and for all the parties and hangouts we made. Fondly referred to as the "Tea, Coffee and Snacks" lab by Prof. C. Pandu Rangan, I am thankful for being a part of it all these years. In particular, I would like to thank Suvradip Chakraborty, Arinjita Paul, Shreyas Shetty, Amit Rawat, Manas Jyoti Kashyop, Rajesh Pandian, Vijayaraghunathan Ramamoorthi, Anshu Yadav, Rajarshi Biswas, Rachit Garg, Kanika Gupta for being a part of my PhD journey. Of special mention is Suvradip Chakraborty. He was a PhD student one year senior to me and we grew up to be very good friends from several enjoyable technical discussions as well as thought-provoking interactions we had on matters about life.

IIT Madras (and Chennai) is the place where I got the first experience of a hostel life. When I joined here in 2015, the only friend I had (as well as the only person I knew) in Chennai was Sankardeep Chakraborty. He was pursuing PhD at IMSc Chennai at that time. I would like to thank him for all the help about various social and academic aspects during my initial days here. Apart from lab-mates, I met many new souls from other labs and departments some of whom are good friends now. Specifically, I would like to acknowledge Anurag Swarnkar, Rahul Bhattacharya and Suchetana Gupta for their help with various technical and non-technical issues during my period of stay here.

The IIT campus had stunned me from the very first day with its thriving nature. To this day, I still get amazed as to how such a beautifully serene campus with such rich flora and fauna can exist amidst quite a busy place in Chennai. I am deeply grateful to all the twists of fate that landed me up here in Chennai and IIT Madras and for the major events that followed. The first such event that affected my life positively was when my wife, Ananya joined IIT Madras for PhD in 2016. It was a joyful moment and possibly one of the best things that could have happened for both of us. Despite going through her world of experiences of an entirely different kind of research, I am extremely grateful for her enormous support at all levels with everything I am doing today. Our exploration together, of southern India in terms of the lush greenery it offers as well as the cultural vibrancy and spiritual ethos of the place has etched several memories that we will cherish forever. Secondly, I am deeply indebted to the book by Swami Vivekananda on the path of Raja yoga (for the profound science of what it is) and Sadhguru Jaggi Vasudev (and Isha foundation) for introducing me to this path through some initiatory practices. On top of that, the tranquillity of the IIT Madras campus has helped me to reflect inward for which I would forever be grateful to this place.

Finally, yet importantly, this section would be vastly incomplete without acknowledging my family – my parents as well as parents-in-law, my brother and brother-in-law. Their endless love and constant encouragement for all these years have motivated and shaped me greatly to what I am today. In particular, not enough can be said about my mother. Aside from being a very broad-hearted person, she has put tireless efforts in raising us to push our way forward in whatever we wanted to do. The same holds for my father, he being the backbone of the social and financial support that my mother has been able to provide us. Any number of words will always fall short of expressing my reverence towards them.

ABSTRACT

KEYWORDS: Functional encryption, Attribute based encryption, Uniform models of computation, Turing machine, Finite automata, LWE, DLIN

The rise of cloud computing and big data technologies in the last two decades has facilitated the tasks of data storage, sharing and outsourcing computations enormously. In particular, these technologies have a tremendous number of applications in the spheres of healthcare, education, transportation, social media and entertainment, and such others. However, sharing sensitive data in the clear puts the clients' trust implicitly on the service provider. This has led to several real time security breaches such as [Cambridge-Analytica, (2018); Doe (2019)]. In fact, a recent survey by [PTI, (2019)] found that around five hundred Indian organizations suffered a whopping loss of Rs. 12.8 crore due to data breaches between July 2018 and April 2019. The situation is aggravated to such an extent that security experts are now predicting cyber-security trends and possible threats that the world is going to witness in 2019 and beyond [Vishwanath (2019); Thompson and Trilling, (2018)]. To curb this menace while retaining the benefits of such technologies, an urgent treatment is needed which this thesis attempts to address in part.

In this thesis, we study methods of encrypting data that still permits service providers to perform authorized computations over it, while leaking no other information about the underlying data. This leads us to study more "expressive" cryptographic primitives like functional encryption (FE) and attribute based encryption (ABE) as generalizations of traditional public key encryption (PKE). In practice, data comes with arbitrary size. Therefore, we represent algorithms operating over the encrypted data with uniform models of computation like Turing machines and finite automata.

The overall thesis can be broadly viewed in two parts.

In the first part, we develop new techniques to construct FE and its generalization to multi-input FE (and the related primitive of indistinguishability obfuscation (iO)) for Turing Machines generically, from minimal assumptions. Our FE and multi-input FE constructions support unbounded length inputs and achieve optimal parameters. All the constructions overcome the barrier of sub-exponential loss incurred by all prior work. Further, our techniques are new and from first principles that avoid usage of sophisticated iO specific machinery used by all relevant prior work.

In the second part, we develop new techniques to construct ABE schemes supporting finite automata based on concrete assumptions from lattices and bilinear maps.

In particular, we construct the first symmetric key ABE scheme for nondeterministic finite automata (NFA) from the learning with errors (LWE) assumption over lattices. Our scheme supports unbounded length inputs, unbounded length machines and unbounded key requests. Further, we leverage our ABE to achieve (restricted notions of) attribute hiding analogous to the circuit setting, obtaining the first predicate encryption (PE) and bounded key FE schemes for NFA from LWE. We achieve machine hiding in the single/bounded key setting to obtain the first reusable garbled NFA from standard assumptions. In terms of lower bounds, we show that secret key FE even for deterministic finite automata (DFA), with security against unbounded key requests implies iO for circuits; this suggests a barrier in achieving full fledged FE for NFA.

We continue in the same thread to study ABE for DFA. Here, we construct the first ABE scheme for DFA from a standard, static assumption on pairings, namely, the decisional linear (DLIN) assumption. Our scheme supports unbounded length inputs, unbounded length machines and unbounded key requests. Our techniques are at least as interesting as our final result. We present a simple compiler that combines constructions of unbounded ABE schemes for monotone span programs (MSP) in a black box way to construct ABE for DFA. Our construction uses its building blocks in a symmetric way – by swapping the two crucial underlying tools of an unbounded key-policy and an unbounded ciphertext-policy ABE for MSPs we also obtain a construction of ciphertext-policy ABE for DFA from static assumptions for the first time. At the heart of our work is the observation that unbounded, multi-use ABE for MSP already achieve most of what we need to build ABE for DFA.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOTATION

Below, we summarize the common notations and definitions used across the thesis.

1. Vectors and matrices are denoted by bold-faced small letters (eg., $\mathbf{a}, \mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}$ etc.) and capital letters (eg., $\mathbf{A}, \mathbf{B}, \mathbf{D}, \mathbf{I}, \mathbf{W}, \mathbf{V}$ etc.) respectively.
2. A function $\mathrm{poly}(n)$ is called a *polynomial* function in n if it is $O(n^c)$ for some constant $c > 0$.
3. A function $\mathrm{negl}(n)$ is called a *negligible* function if it is $O(n^{-c})$ for all $c > 0$. An event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$.
4. *Probabilistic polynomial time* is abbreviated as PPT.
5. The function $\log x$ is the base 2 logarithm of x.
6. For any finite set S, $\mathcal{P}(S)$ denotes the power set of S.

CHAPTER 1

Introduction

1.1 Motivation

The internet of today's world contains a plethora of applications like e-commerce, social networks, online banking, crowdfunding, multiplayer online games and so on. The rise of cloud computing and big data management has led to all these applications resulting in massive amounts of information stored in physical servers that are located remotely. Although most of these applications have turned out to be quite beneficial, privacy breaches are abound in these contexts. Therefore, a central challenge in cryptography today is how to retain the benefits of all these applications while safeguarding privacy and security. This prompts us naturally to seek sound methods for controlling disclosure to sensitive information while performing authorized computation over it. Towards addressing this challenge, this thesis develops secure methods for performing computations over encrypted data. In particular, we study the notion of functional encryption (FE), which is a generalization of traditional public-key encryption (PKE) [Diffie and Hellman, (1976); Rivest *et al.* (1978)], as explained below.

In the conventional PKE setting, any user, say Alice, generates a public and private key pair that are mathematically related. As the name suggests, the public key is announced to everyone and the secret key is kept private. A user who wants to communicate with Alice, say Bob, will encrypt his message using Alice's public key to send her a ciphertext. On receipt of this ciphertext, Alice can use her secret key to decrypt and read Bob's message. While this technology has been widely accepted and deployed successfully for over two decades now, it remains insufficient for modern purposes. Let us understand this with a toy example.

Bob needs to share a large set of files containing confidential information with some of the users of his choice in a network. Additionally, he also wants to enforce an access control mechanism so that any user will be able to decrypt and read only those files which they are individually authorized to access.

Employing PKE to solve this problem means Bob needs to encrypt a subset of files from this collection under separate public keys corresponding to individual users with whom he wants to share this set. So, there is a replication of the same message being encoded as different ciphertexts. Even if Bob needs to give access to two distinct subsets of files without any intersection, to two different users, he still needs to encrypt each subset separately under the appropriate public key of the respective user. This provides a highly inefficient solution. Abstracted out, this problem stems from the fact that the decryption in any PKE system follows an "*all-or-nothing*" paradigm, i.e.,

- Any user with a valid secret key can always learn the whole underlying plaintext (the entire collection in this case) by the functionality of PKE.
- Users, not in possession of a valid secret key, can learn nothing about the plaintext (except its length) by the security of PKE.

Hence, the notion of PKE is insufficient for such applications.

Faced with the above goals, it is natural to ask if there is a way to perform fine-grained access control mechanisms over encrypted data. Note that a whole body of theory and its implementation have already been developed and deployed for access control mechanisms over databases. In this case, it is only required to store the database in plaintext form. When the database is kept and maintained in a third-party cloud server for storage issues, these access control mechanisms fall short of retaining database privacy. So, the question that we ask here is whether we can first encrypt the database to store as ciphertext in the cloud and later hope to perform selective computation over this encrypted database. This question gave birth to the area of functional encryption in the work of [Sahai and Waters (2005)]. Later, [Boneh *et al.* (2011); O'Neill (2010)] formalized the notion of FE as a generalization that encompasses a broad view of encryption systems like Identity-based Encryption (IBE), ABE and Predicate Encryption (PE). FE generalizes even PKE in the sense that it allows secret keys to correspond to "*functions*" rather than users. For example, a ciphertext can encode the database of grades for all students who appeared in an examination conducted across various physical locations countrywide. In this setting, a function secret key could correspond to an "Average" function, so that decryption reveals only the average grade of all the students. The security goal of such a system aims to prevent any further information leakage about the database other than what is already revealed from the function and its output. It is easy to see that this kind of encryption system helps to solve the toy example described before. Bob simply encrypts his collection of files as a whole and publishes this ciphertext. The secret key will correspond to a function that takes the collection of files as input and outputs only a subset of files as per its description. Any user in possession of such a secret key can decrypt and learn the contents of only those files in the collection that it was authorized to know individually.

The three major aspects of any cryptographic primitive are its functionality, security, and efficiency. For FE schemes, the supported class of functions is evidently an important parameter. In particular, representing a class of functions meaningfully, for which function secret keys need to be generated, plays a pivotal role in the practical usefulness of the primitive. The theory of computing represents functions or algorithms via models of computation that can be divided broadly between two classes, *non-uniform* and *uniform*. We describe them below briefly.

Non-uniform models of computation: The description of such a model depends inherently on the input length. Hence, the same functionality handling two different input lengths have different descriptions. Boolean circuits are a widely studied example of such a model. Here, the computation is executed by a bunch of basic logic gates from $\{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}\}$ arranged in a certain topological order demanded by the requirement of the function description. In terms of graphs, any circuit is represented via a directed acyclic graph (DAG) [Arora and Barak (2009)] consisting of *input* vertices (with in-degree = 0) labelled with literals taking values from $\{0, 1\}$ and *internal* vertices labelled by Boolean operators from $\{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}\}$. For each internal vertex, values on its incoming edges are examined and then the boolean operation that this node is labelled with, is applied on those values. This assigns a value to each node propagating according to a breadth-first search (BFS) traversal on the DAG, until the last step where the value assigned to the final vertex is taken as the circuit output.

Uniform models of computation: As hinted from the name, the description of this computation model is independent of any input length. Hence, a single, fixed description of such a model corresponding to a function can handle inputs of arbitrary length. Examples of such models include finite automata (FA), Turing machine (TM) and random access machine (RAM). In these models, any computation is executed as dictated by a *transition function* that forms the core component in the description of these machines [Sipser (1996); Hamlin *et al.* (2019)]. In particular, the transition function dictates the states that the computation can be in at any time step after reading (and/or writing, in case of TM and RAM).

While circuits are a powerful model of computation, they suffer from two major drawbacks. The first one, as already described above, is that circuits only support fixed input lengths and that their description changes based on different input lengths. Real-world, large-scale data rarely come with an apriori known bound on its length. On top of that, FE systems where functions are represented as circuits, must also supply different secret keys for the same function, if the input lengths vary[1]. Secondly and more importantly, circuits always incur *worst-case* runtime over all inputs of a fixed length. In particular, any uniform model like FA, TM or RAM can be converted into a circuit essentially by unrolling loops and accounting for all branches of the underlying computation. But this transformation results in the worst-case runtime over all inputs of a fixed length which makes any application inherently slow in practice, even if there were no cryptographic overheads to deal with. Ideally, we want an algorithm to run in time depending on the exact input that it has been fed with. We call such a runtime as *input-specific* runtime that we cannot hope to obtain for circuits. In general, such a limitation on non-uniform computation poses an implicit and severe barrier in the practical deployment of computations on encrypted data. On the flip side, none of these disadvantages is there when we represent functions as uniform computation models. Specifically, they have fixed descriptions that can handle inputs of any length. Hence, FE systems supporting such models of computation have to issue only a single secret key per functionality that would work with any input length. Further, they work akin to real-world algorithms and programs incurring input-specific runtime. Therefore, while designing FE systems, we would like to model the underlying functions as a uniform model which directly impacts the efficiency of such systems.

Modern cryptography leverages security from the conjectured intractability of some mathematical problems. The formal way to argue security is to construct a polynomial time reduction to some problem with conjectured hardness, from the cryptographic primitive in context. This reduction is modelled as a game (or experiment) between two imaginary entities (technically, two randomized polynomial time algorithms) called "challenger" and an "adversary". The adversary wishes to break the primitive while

[1]A trivial workaround would be to fix the input length to some fixed upper bound and pad all data to this bound; but this solution incurs substantial overhead (besides being inelegant).

the challenger has a dual role to play. On one hand, it runs the cryptographic scheme honestly to simulate adversary's view of the system. At the same time, it also encodes an instance of some hard problem cleverly within the components that the adversary gets to see in its view. Ensuring this is the case, it uses the adversary's output to construct a solution to the underlying problem instance, that is conjectured to be hard. This way we obtain provable security guarantees for the primitive at hand.

From its advent till now, modern cryptography continues to evolve with solid foundations as well as several applications. All of it (except a few primitives that aim for *unconditional* security) comes with an underlying bunch of computational assumptions on which security relies on. Today such computational assumptions range from widely believed ones like factoring of integers, computing discrete logarithm in certain groups, various assumptions on bilinear maps, finding short vectors in lattices and such others to some new, non-standard ones like reliance on multilinear maps, indistinguishability obfuscation (iO) and the like. Naturally, we desire to build cryptography based on weaker and well-understood hardness assumptions. To this end, the work done in this thesis constructs new FE and ABE schemes for TM and FA models of computation respectively from *weaker* assumptions. In particular, all our constructions rely on cryptographic assumptions that are weaker than the ones that existing constructions relied on in the literature. We now present a brief overview of the thesis below.

1.2 Overview of the thesis

We start with studying FE schemes supporting the TM model of computation. In this context, we should first elucidate a bit on the related notion of iO here. The notion of iO, originating in the work of [Barak *et al.* (2001)], seeks to garble programs such that the obfuscations of any two functionally equivalent programs are indistinguishable. While non-obvious at first what such a guarantee is good for, iO has emerged as a surprisingly powerful notion in cryptography, leading to many advanced cryptographic applications that were previously out of reach [Garg *et al.* (2013*a*); Sahai and Waters (2014); Canetti *et al.* (2015, 2014); Bitansky *et al.* (2015*a*); Koppula *et al.* (2015); Bitansky *et al.* (2015*b*); Lin *et al.* (2016); Cohen *et al.* (2016); Carmer *et al.* (2017); Liu and Zhandry (2017)]. The list of applications of iO shown here is not at all exhaustive and continues to grow even today. It is well known that FE for circuits has been shown to imply iO for circuits, albeit with sub-exponential loss [Ananth and Jain (2015); Bitansky and Vaikuntanathan (2015)]. Over the last few years, both FE and iO have received significant attention, with a rich body of work that attempts to support uniform models of computation [Bitansky *et al.* (2015*a*); Canetti *et al.* (2014); Koppula *et al.* (2015); Chen *et al.* (2015*b*); Canetti *et al.* (2016); Ananth *et al.* (2016); Canetti and Holmgren (2016)], rely on weaker assumptions [Brakerski *et al.* (2016); Garg and Srinivasan (2016); Li and Micciancio (2016); Bitansky *et al.* (2016); Komargodski and Segev (2017); Ananth and Sahai (2017); Lin (2017); Lin and Tessaro (2017); Kitagawa *et al.* (2017, 2018*a*,*b*)], achieve stronger security [Ananth *et al.* (2015*a*); Brakerski *et al.* (2016)] and greater efficiency [Ananth *et al.* (2017)]. In our work in [Agrawal and Maitra (2018)], we make further progress towards the goal of basing both FE and iO on *minimal* assumptions, in the TM model of computation. This question has been studied extensively [Goldwasser *et al.* (2013*a*); Ananth and Sahai (2016); Bitansky *et al.* (2015*a*); Canetti *et al.* (2014); Koppula *et al.* (2015); Chen *et al.* (2015*b*); Canetti *et al.* (2016); Ananth *et al.* (2016); Canetti and Holmgren (2016); Ananth *et al.* (2017)] - we refer an interested reader to [Ananth and Sahai (2016); Ananth *et al.* (2017)] for a detailed discussion. We construct FE and iO for TMs from the minimal assumption of compact FE for circuits. In particular, our constructions overcome the barrier of sub-exponential loss incurred by all prior work.

Next, we study ABE schemes supporting the FA model of computation. Let us start with a high-level view of what ABE as an emerging paradigm of encryption aims to achieve. ABE [Sahai and Waters (2005); Goyal *et al.* (2006)] comes in two different modes, namely ciphertext-policy ABE (CPABE) and key-policy ABE (KPABE). In a CPABE system, a ciphertext of a message m is labelled with a Boolean function f and secret keys are labelled with public attributes $\mathbf{x}$. Alternately, in a KPABE system the role of f and $\mathbf{x}$ gets swapped, i.e., a ciphertext embeds a public attribute $\mathbf{x}$ while secret keys embed Boolean functions f. In both cases decryption succeeds to yield the hidden message m if and only if the attribute satisfies the function, namely $f(\mathbf{x}) = 1$. Constructing ABE for uniform models of computation from standard assumptions, is an important problem, about which very little is known. The only known ABE schemes in this setting that i) avoid reliance on multilinear maps or indistinguishability obfuscation, ii) support unbounded length inputs and iii) permit unbounded key requests to the adversary in the security game, are by [Waters (2012)] and its variants. Waters provided the first ABE for deterministic finite automata (DFA) satisfying the above properties, from a parametrized or "q-type" assumption over bilinear maps. Generalizing this construction to nondeterministic finite automata (NFA) was left as an explicit open problem in the same work, and has seen no progress to date. Although we know of constructions from non-standard assumptions [Ananth and Sahai (2017); Agrawal and Maitra (2018); Kitagawa *et al.* (2019)] for the more general class of TMs, constructions from other concrete assumptions such as more standard pairing based assumptions, or lattice based assumptions has also proved elusive. In our work in [Agrawal *et al.* (2019*a*)], we construct the first symmetric key KPABE scheme for NFA and its generalizations to PE, bounded key FE and reusable garbling scheme from the learning with errors (LWE) assumption in lattices. We also illuminate a barrier in generalizing further to obtain fully collusion-resistant symmetric key FE in the sense that it would imply iO for circuits.

We continue in the same thread as before but turn our focus towards ABE schemes for DFA from assumptions on bilinear maps. As mentioned earlier, the only other full-fledged KPABE construction for DFA in the literature before 2019 was by [Waters (2012)], which had to rely unfortunately on a parametrized assumption over bilinear maps. Obtaining a construction from *static* assumptions has been elusive, despite much progress in the area of ABE in general. In our work [Agrawal *et al.* (2019*b*)], we construct the first ABE scheme for DFA from static assumptions on pairings, namely, the decision linear (DLIN) assumption. We obtain new techniques to simulate DFAs through monotone span programs (MSP) and use existing constructions of some special ABE schemes for MSP. Our work also leverages a technique from [Agrawal *et al.* (2019*a*)] to construct ABE that support unbounded DFA machines and unbounded inputs by combining ABE schemes that are bounded in one co-ordinate. Further, our construction is a generic compiler and yields both KPABE and CPABE schemes for DFA from the DLIN assumption. We note that CPABE for DFA from standard, static assumptions was an open problem and our work is the first one which addresses and constructs it.

1.3 Organization

The thesis is organized in five chapters. Chapter 1 consists of the motivation and an overview of the thesis. The prerequisite definitions that are needed to understand the work will be introduced in the respective chapters as and when required. In Chapter 2, we describe our results for FE in the TM model. Chapters 3 and 4 describes our results on ABE for NFA and DFA respectively. We conclude the thesis in Chapter 5. Appendices A, B and C contain the supplementary material for Chapters 2, 3 and 4 respectively.

Remarks.

1. We clarify the difference between homomorphic encryption (HE) and FE and ABE as follows, as asked by a reviewer. Homomorphic encryption is a paradigm that allows to compute a function *homomorphically* over encrypted data. The result of such computations are ciphertexts which do not reveal even the function output of the underlying data to a user who does not posses a secret key for decryption. On the contrary, the work done in this thesis is on FE and ABE schemes. Both ABE and FE systems allow to generate *constrained* secret keys which can decrypt ciphertexts to reveal information about the plaintext *in the clear*. Compared to these systems, HE schemes allow to compute only ciphertexts encoding function outputs and reveals nothing about the original plaintexts as well as function outputs.

2. The initial thesis title, “*Encrypted Computation for Uniform Models*" has been changed to “*Fine-grained Encryption for Uniform Models of Computation*" as per one of the reviewer’s suggestion. We discuss the reason for this change in title below.

   The initial title aimed to highlight the research theme succinctly, which is to study computing functions on encrypted data when they are represented by uniform models of computation. As pointed by one of the reviewers, the main contents of Chapters 3 and 4 focuses on attribute-based encryption, where the computing is done on a public attribute and not on encrypted data (with the exception in Chapter 2). Note that we generalize our ABE scheme for finite automata to support *predicate encryption*, *bounded-key functional encryption* and *reusable garbling* variants, where the attribute is indeed hidden on which the computation is done (except in Chapter 4, where we construct ABE only for deterministic finite automata). But, since these generalizations are presented in the Appendices only, we have changed the title as per suggestion, as it fits better with the main content of the thesis.

3. In Chapter 2, we construct FE, multi-input FE and iO for Turing machines from the minimal assumptions of compact, polynomially secure FE and sub-exponentially secure FE for circuits respectively. In this context, *our* choice of the word “*minimal*" stems from the following facts:

   1. Our FE and iO for Turing machines constructions avoid the sub-exponential loss barrier incurred by all prior work, thus improving the security assumptions required to build them from compact FE for circuits only.

   2. Our multi-input FE for Turing machines construction (restricted to the bounded arity setting) relies only on sub-exponentially secure FE for circuits. Prior to our work, we only knew such a construction from public coin differing inputs obfuscation (which is a strong knowledge type assumption).

   In essence, we get *asymptotically tight* reductions for FE, multi-input FE, and iO for Turing machines from the same assumptions as needed for circuits.

CHAPTER 2

Functional Encryption and Indistinguishability Obfuscation for Turing Machines from Minimal Assumptions

2.1 Introduction

In this chapter, we study functional encryption (FE), multi-input FE (miFE) and indistinguishability obfuscation (iO) in the Turing machine model of computation.

Functional encryption (FE) [Sahai and Waters (2005); Boneh *et al.* (2011); O'Neill (2010)] is a generalization of public key encryption that enables fine grained access control on encrypted data. In FE, a secret key corresponds to a function f and ciphertexts correspond to strings from the domain of f. Given a function secret key SK_f and a ciphertext $\mathsf{CT}_{\mathbf{x}}$, the decryptor learns $f(\mathbf{x})$ and nothing else. FE can be naturally generalized to a multi-input setting where there are multiple parties, say $p_1, \ldots, p_n$ encrypting their own messages $\mathbf{x}_1, \ldots, \mathbf{x}_n$. A function secret key now corresponds to function f of arity n. As before, given SK_f and ciphertexts from n parties, namely $\mathsf{CT}_{\mathbf{x}_1}, \ldots, \mathsf{CT}_{\mathbf{x}_n}$, the decryptor learns $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and nothing else.

Program obfuscation [Barak *et al.* (2001)] aims to alter a program into an unintelligible one such that its functionality is still preserved. [Barak *et al.* (2001)] showed that the most general notion of such obfuscation is impossible to construct for all Turing machines. Faced with this impossibility, they proposed the notion of indistinguishability obfuscation (iO). iO seeks to garble programs such that the obfuscation of any two functionally equivalent programs of roughly the same size are indistinguishable. FE has been shown to imply iO, albeit with sub-exponential loss by [Ananth and Jain (2015); Bitansky and Vaikuntanathan (2015)].

Our work provides new constructions of FE, miFE and iO for Turing machines from minimal assumptions. Before stating our contributions in detail, we first summarize below the state of art prior to our work:

1. For *single input* FE for TMs that accept unbounded length inputs and place no restriction on the description size or space complexity of the machine, the state of art was the work of [Ananth and Sahai (2016)], which relies on the existence of iO for circuits.

2. For *multi-input* FE in the TM model, the only known construction till now is [Badrinarayanan *et al.* (2015)], which relies on the existence of public coin differing inputs obfuscation (diO).

3. iO for Turing Machines with unbounded memory and bounded inputs are constructed in the works of Koppula et al. and Ananth et al. [Koppula *et al.* (2015); Ananth *et al.* (2017)]. Both works rely on the existence of sub-exponentially secure iO for circuits along with other standard assumptions. We note that FE for circuits implies iO with sub-exponential loss, so when relying on FE for circuits, these works incur sub-exponential loss from two sources.

2.2 Our Contributions

An FE scheme for circuits is called *compact* if the running time of the encryption algorithm depends only on the security parameter and the input. We denote such a scheme by CktFE. We construct FE, miFE and iO for TMs from the *minimal* assumption of compact FE for circuits. Our constructions overcome the barrier of sub-exponential loss incurred by all prior work. Our contributions are the following.

1. We provide a new construction of single input FE for TMs with unbounded inputs, achieving optimal parameters from *polynomially* secure, compact FE for circuits. The previously best known construction by [Ananth and Sahai (2016)] relies on iO for circuits, or equivalently, sub-exponentially secure FE for circuits. We note that iO for circuits implies decomposable compact FE for circuits [Garg *et al.* (2013*a*)] (please see Appendix A.5), so our construction also implies FE for TMs from iO for circuits.

2. We provide a new construction of multi-input FE for TMs. Our construction supports a fixed number of encryptors (say k), who may each encrypt a string $\mathbf{x}_i$ of *unbounded* length. We rely on sub-exponentially secure FE for circuits, while the only previous construction [Badrinarayanan *et al.* (2015)] relies on a strong knowledge type assumption, namely, public coin differing inputs obfuscation. The arity k supported by our scheme depends on the underlying multi-input CktFE scheme, for instance using [Komargodski and Segev (2017)], we can support $k = \mathsf{polylog}(\lambda)$.

3. We construct iO for TMs with bounded inputs and unbounded memory from the same assumptions as required by iO for circuits, namely, sub-exponentially secure FE for circuits. The previous best constructions [Koppula *et al.* (2015); Ananth *et al.* (2017)] require sub-exponentially secure iO for circuits, which in turn requires sub-exponentially secure FE for circuits [Ananth and Jain (2015); Bitansky and Vaikuntanathan (2015)], resulting in double sub-exponential loss.

Our constructions make use of FE for circuits that satisfy a mild property called *decomposablity*, which we show how to construct generically from FE for circuits. Decomposable FE, analogously to decomposable randomized encodings [Applebaum *et al.* (2014)], roughly posits that a long string be encrypted bit by bit using shared randomness across bits. This property is already satisfied by all known constructions of CktFE in the literature to the best of our knowledge, please see Appendix A.5.1.

Our techniques are new and from first principles, and avoid usage of sophisticated iO specific machinery such as positional accumulators and splittable signatures that were used by all prior work [Koppula *et al.* (2015); Ananth and Sahai (2016); Ananth *et al.* (2017)]. Our work leverages the security notion of distributional indistinguishability (DI) for CktFE which was first considered by [Gentry *et al.* (2014)], who provided a construction for single input FE satisfying DI security assuming the existence of iO. We strengthen this result by constructing DI secure CktFE from standard CktFE. Please see Figure 2.1 for an overview of our results.

2.3 Additional Prior Work

Since iO is considered an inherently sub-exponential assumption and much stronger than the polynomial assumption of compact FE, replacing iO by FE in cryptographic constructions has already been studied extensively, for instance in the context of PPAD hardness [Garg *et al.* (2016)], multi-input FE for circuits [Brakerski *et al.* (2016); Komargodski and Segev (2017)] as well as trapdoor one-way permutations and universal samplers [Garg *et al.* (2017)]. We note that aside from reliance on weaker, better understood assumptions, avoiding sub-exponential loss results in significantly more efficient schemes. We refer the reader to [Garg *et al.* (2017)] for a detailed discussion.

Distributional indistinguishability was also considered in the context of output compressing randomized encodings Lin *et al.* (2016); indeed, this work implies that achieving

DI security for FE for Turing machines with *long* outputs is impossible in the plain model. We note that our construction sidesteps this lower bound by considering Turing machines with a single output bit.

iO for TMs with unbounded memory has been constructed by [Koppula *et al.* (2015); Ananth *et al.* (2017)] as discussed above, other prior works were limited to bounded space constraints. We note that [Ananth *et al.* (2017)] additionally achieve constant overhead in the size of the obfuscated program as well as amortization, which we do not consider in this work. We also note that the work of [Badrinarayanan *et al.* (2015)] achieve miFE for TMs where the number of encrypting parties can be arbitrary, whereas we only support a-priori fixed, bounded number of parties.

The approach of using decomposable FE for circuits to construct FE for deterministic finite automata (DFA) in the *single key* setting was suggested by [Agrawal and Singh (2017)]. In this work we develop and significantly generalize their ideas. In particular, we handle the unbounded key setting in FE for TMs which necessitates dealing with the much more complex indistinguishability style definition, for which we develop new proof techniques which use a novel "sliding trapdoor" approach and leverage distributional indistinguishability. In contrast, since [Agrawal and Singh (2017)] use simulation security for single key FE, their proof must not contend with any of these challenges. Please see below for details.

2.4 Our Techniques

We describe an overview of our constructions, starting with single input FE, generalizing to multi-input FE and then building iO. All our constructions support the Turing machine model of computation. Our constructions rely on a single input FE scheme for *circuits*, denoted by CktFE, which satisfies *decomposability*. We will also show later in Appendix that decomposable FE for circuits is implied by FE for circuits. Intuitively, decomposability means that the ciphertext $\mathsf{CT}_{\mathbf{x}}$ for a multi-bit message $\mathbf{x}$ be decomposable into multiple ciphertext components CT_i for $i \in |\mathbf{x}|$, one for each bit x_i of the message. Moreover, the ciphertext components encoding individual bits of a single input are tied together by common randomness, that is $\mathsf{CT}_i = \mathcal{E}(\mathsf{PK}, r, x_i)$ where $\mathcal{E}$ is an encoding

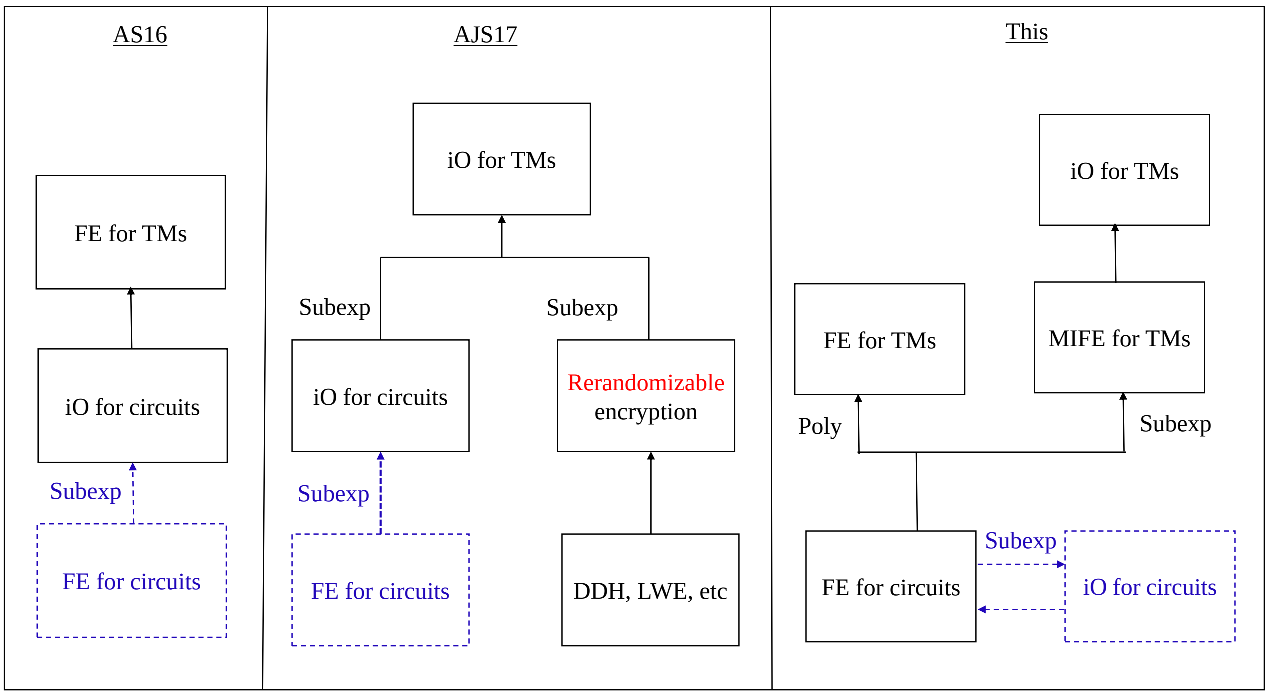

Figure 2.1: Prior work and our results. The reductions with subexponential loss are specified, no specification implies standard polynomial loss. The dashed blue lines indicate primitives that are not actually used by the work in question; we add these to elucidate the relationship between primitives. We do not include [Badrinarayanan *et al.* (2015)] here since it relies on public coin diO.

function and r is common randomness used for all $i \in |\mathbf{x}|$[1]. The notion of decomposability has been widely studied and used in the context of randomized encodings, which may be seen as a special case of functional encryption; please see [Applebaum *et al.* (2014)] as an example. We note that all known FE schemes in the literature are already decomposable to the best of our knowledge, please see Appendix A.5.1 for a discussion.

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

Our construction uses two CktFE schemes, $1\mathsf{FE}_1$ and $1\mathsf{FE}_2$, where $1\mathsf{FE}_2$ is decom-

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

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

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

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

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

Decomposability. The above construction relies on the underlying CktFE scheme satisfying the property of decomposability. As already mentioned before, decomposability is a mild assumption and already satisfied by all known CktFE constructions in the literature to the best of our knowledge (please see Appendix A.5.1 for a discussion). We can also remove the requirement of decomposability with the minimal assumption of one-way functions but at the expense of making our compiler more complicated[2]. However, a cleaner approach is to build decomposable FE generically from standard FE, by using decomposable randomized encodings, which may be constructed from one way functions. Please see Appendix A.5 for details.

Encoding the PRF key. The above informal description hides an important detail – for the function key to produce ciphertext components using a PRF, it must have the key of the PRF, chosen by the encryptor[3], passed to it as input. Thus the ciphertext must additionally encode the PRF key along with inputs (t, x, q). However, the ciphertext is constructed using randomness derived from the same PRF- resulting in circularity. We resolve this difficulty by using *constrained* PRFs [Boneh and Waters (2013); Kiayias *et al.* (2013); Boyle *et al.* (2014)], and having a ciphertext encode a PRF key that only allows computation of randomness for time steps *of the future*; this does not compromise its own security. For this constraint family, we provide a construction of cPRFs from one-way functions. We believe this construction and the method of its application may be useful elsewhere[4].

More formally, our construction makes use of constrained, delegatable PRF for the

[2]Intuitively, we use decomposability because the “symbol" and “state" components of the ciphertext are generated during different times in decryption, say T_1 and T_2. However, since the underlying CktFE is compact, generating longer outputs comes for free. Hence, we can have the CktFE generate the complete (symbol, state) CT for the relevant symbol and all possible states at time T_1. Given in the clear, this would be insecure but this can be fixed by further nesting these CTs within a symmetric key encryption scheme and outputting them (in randomly permuted order). Later, at time T_2, when the state is computed, the decryption can output the SKE key to unlock the appropriate CktFE CT.

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

[4]For instance, a similar situation w.r.t circularity arises in the original garbled RAM construction of Lu and Ostrovsky [Lu and Ostrovsky (2013)].

function family $f_t : \{0,1\}^{2\cdot\lambda} \to \{0,1\}$ defined as follows.

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

We denote the constrained PRF key K_{f_t} by K_t for brevity. By the delegation property of constrained PRFs, we have that if $t' \geq t$ then $\mathsf{K}_{t'}$ can be derived from K_t. The proof requires the PRF to be punctured at a fixed point in each hybrid, we provide a construction of delegatable punctured PRF in Appendix A.3.

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

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

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

A standard approach to deal with this difficulty is to embed a "trapdoor" mode within the functionality [Ananth *et al.* (2015*a*); Ananth and Jain (2015); Brakerski *et al.* (2016)] which lets us "hardwire" the ciphertexts that must be output by decryption directly in the key, allowing decryption to yield an inconsistent chain. However, this approach also fails in our case, since the length of the chain is unbounded and there isn't sufficient space in the key to incorporate all its values.

Our Approach: "Sliding" Trapdoors. We deal with this difficulty by designing a novel "sliding-window" trapdoor approach which lets us hardwire the decryption chain "piece by piece". In more detail, we start with the last two time steps $(T, T-1)$, program the key to produce the output corresponding to $b = 1$ for time step T and $b = 0$ for $T-1$, then transition to a world where the output corresponds to $b = 1$ for both T and $T-1$. At this point, the hardwiring of the output for time step T is redundant, since the ciphertext output by the decryption process at time step $T-1$ automatically computes the output coresponding to $b = 1$ at time step T. Thus, we may now *slide* the trapdoor to program to the next pair $(T-1, T-2)$, switching the decryption output at time step $T-2$ to $b = 1$ and so on, until the entire chain corresponds to $b = 1$.

Intuitively, we are "programming" the decryption only for outputs at both ends of the "broken link", so that preceding links are generated using $b = 0$ and subsequent links are generated using $b = 1$. We leverage the fact that the chain links corresponding to future time-steps are encoded *implicitly* in a given time step – hence if we manage to hide the chain inconsistency at a certain position i, this implies that the remainder of the chain is constructed using the bit encoded at step i. Formalizing this argument requires a great deal of care, as we must keep track of the "target" time steps corresponding to the two ends of the broken link that are being programmed, the time steps at which the symbol and state ciphertexts are generated to be "consumed" at the target time-steps, the particular values that must be encoded in the symbol, state fields in both cases as well as the key that is being handled at a given time in the proof. This technique allows us to obtain a proof for selective security for the single input TMFE scheme. We defer the details to the full proof in Section 2.7.3.

Generalising to Multi-Input FE for Turing machines. For the k party setting, a natural idea is to have each party encrypt its own input $\mathbf{x}_i$, and use a k input CktFE scheme kFE [Brakerski *et al.* (2016); Komargodski and Segev (2017)], to "aggregate" these into the "input" ciphertext $\mathsf{CT}(\mathbf{x})$ for one long input $\mathbf{x} = (\mathbf{x}_1 \| \mathbf{x}_2 \| \ldots \| \mathbf{x}_k)$, under a different CktFE scheme $\mathsf{1FE}$. Note that the length of $\mathbf{x}$ is unknown hence it may not be encoded "all at once" but must be encoded bit by bit as in the previous scheme. Now, by additionally providing the $\mathsf{1FE}$ ciphertext encoding the start state of the Turing machine $\mathsf{CT}(\mathsf{q}_{\mathsf{st}})$, and a function key to compute the transition table of the TM as in the previous scheme, we may proceed with the computation exactly as before.

Formalizing this idea must contend with several hurdles. In the multi-input setting, the i^{th} encryptor may encode multiple inputs and functionality permits "mix and match" of ciphertexts in the sense that any input encoded by party i may be combined with any input encoded by parties $j \in [k],\ j \neq i$. Therefore, if each of k parties encodes T ciphertexts, there are T^k valid input combinations that the TM may execute on. However, when the TM is executing on any input combination, we must ensure that it cannot mix and match symbol, state pairs across different input combinations. Moreover, an encryption for a symbol, state pair produced by some machine M_i should not be decryptable by any machine M_j for $j \neq i$. These issues are handled by careful design of the aggregate functionality to ensure that an execution thread of any input combination by any machine is separate from any other. We prove selective security for this scheme, which extends naturally from the single input case. Please see Section 2.8 for details.

Distributional Indistinguishability. As discussed above, our constructions rely on the security notion of *distributional indistinguishability* (DI) for functional encryption for circuits [Gentry *et al.* (2014)]. Intuitively, this notion says that if the outputs produced by a circuit on two input distributions are merely indistinguishable (as against exactly equal), then the ciphertexts encoding those inputs must also be indistinguishable. We defer the details of how to construct DI secure single input FE from standard FE later. In Appendix A.4 we give a construction of DI secure single input FE from standard FE.

Indistinguishability Obfuscation. Constructing iO for TMs given miFE for TM is straightforward, and adapts the miFE to iO circuit compiler by [Goldwasser *et al.* (2014)] to the TM setting. As in the circuit case, an miFE for TM that supports two ciphertext queries and a single key query suffices for this transformation. Our security proof for miFE for TM is tight and thus this compiler yields iO for TM from sub-exponentially secure FE for circuits rather than sub-exponentially secure iO for circuits.

2.5 Organization

We organize the rest of the chapter as follows. In Section 2.6 we provide the prequisite definitions used by our constructions. In Section 2.7, we provide our construction for single input FE for Turing machines. In Section 2.8, we provide our construction for multi-input FE for Turing machines for any fixed arity k and in Section 2.9 we describe the construction of iO for Turing machines for bounded inputs. Our constructions use constrained PRFs which are instantiated in Appendix A.3 and decomposable FE which is constructed in Appendix A.5.

2.6 Preliminaries

In this section, we recall some necessary definitions that we require to build our results.

2.6.1 Definitions: Turing Machines

We recall the definition of a Turing machine (TM). A TM M is represented by the tuple $(Q, \Gamma, \beta, \Sigma, \delta, \mathsf{q}_{\mathsf{st}}, F)$ 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 of a TM M (i.e. number of steps the head takes) on an input $\mathbf{w}$ by $\mathsf{runtime}(M, \mathbf{w})$.

Oblivious Turing Machines

Our constructions make use of oblivious Turing machines.

Definition 2.6.1 (Oblivious Turing Machine [Pippenger and Fischer, (1979); Impagliazzo (2011)]). 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.

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 [Pippenger and Fischer, (1979)]. 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), a $O(t)$ 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 and Fischer, (1979)] reduces the time required to $O(t \log t)$.

We note that a slightly different definition of OTMs [Arora and Barak (2009)] 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 OTMs corresponding to M_1 and M_2 are exactly the same.

In our construction, the OTM will be provided the input length of the message as an explicit input, and can use this to compute the head movements at any given time step.

2.6.2 Definitions: FE for Circuits

In this section, we define functional encryption for circuits, in both the single and multi-input setting. Our definitions follow a syntax that will be convenient for constructing our schemes in this chapter.

Single Input Functional Encryption for Circuits

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

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

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

Definition 2.6.2 (Correctness)**.** A functional encryption scheme CktFE is correct if for all $\lambda \in \mathbb{N}$, all $f \in \mathcal{F}_\lambda$ and all $x \in \mathcal{X}_\lambda$,

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

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

Definition 2.6.3 (Compactness [Ananth and Jain (2015)])**.** A functional encryption scheme for circuits is said to be compact if for any input message $\mathbf{x}$, the running time of the encryption algorithm is polynomial in the security parameter and the size of $\mathbf{x}$. In particular, it does not depend on the circuit description size or the output length of any function f supported by the scheme.

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

Distributional Indistinguishability for Circuit FE. In this section we define the notion of distributional indistinguishability for functional encryption for circuits. The notion was first defined by [(Gentry *et al.*, 2014, Sec 3.4)] in the context of reusable garbled circuits, i.e. single key functional encryption, but may be generalized to the multi-key setting in a straightforward way. Intuitively, this notion says that if the outputs produced by a circuit on two input distributions are indistinguishable, then the ciphertexts encoding those inputs must also be indistinguishable.

Definition 2.6.4. A functional encryption scheme $\mathcal{F}$ for a circuit family $\mathcal{G}$ is secure in the distributional indistinguishability game, if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter λ:

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

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

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

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

Decomposable functional encryption for circuits In this section, we recall the notion of *decomposable functional encryption* (DFE) defined by [Agrawal and Singh (2017)]. Decomposable functional encryption is analogous to the notion of decomposable randomized encodings [Applebaum *et al.* (2014)]. Intuitively, decomposability requires that the public key PK and the ciphertext $\mathsf{CT}_{\mathbf{x}}$ of a functional encryption scheme be decomposable into components PK_i and CT_i for $i \in [|\mathbf{x}|]$, where CT_i depends on a single deterministic bit x_i and the public key component PK_i. In addition, the ciphertext may contain components that are independent of the message and depend only on the randomness.

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

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

   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. $\bar{\mathcal{E}}(\mathsf{PK}, \mathbf{y}, r) \triangleq \Big(\mathcal{E}(\mathsf{PK}_1, y_1, r, \hat{r}_1), \ldots, \mathcal{E}(\mathsf{PK}_k, y_k, r, \hat{r}_k)\Big)$, where $|\mathbf{y}| = k$. We assume that given the security parameter, the spaces $\mathcal{P},\ \mathcal{R}_1,\ \mathcal{R}_2,\ \mathcal{C}$ are fixed, and the length of the message $|\mathbf{x}|$ can be any polynomial.

Multi-Input Functional Encryption for Circuits

We define the notion of private-key t-input functional encryption for circuits here. Our definition follows that of [Brakerski *et al.* (2016); Komargodski and Segev (2017)].

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

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

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

Definition 2.6.5 (Correctness)**.** A private-key t-input functional encryption scheme $t\text{-}\mathsf{CktFE}$ is correct if for all $\lambda \in \mathbb{N}$, $f \in \mathcal{F}_\lambda$ and all $(\mathbf{x}_1, \ldots, \mathbf{x}_t) \in (\mathcal{X}_1)_\lambda \times \ldots \times (\mathcal{X}_t)_\lambda$,

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

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

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

Definition 2.6.6 (Valid t-Input Adversary)**.** A PPT algorithm $\mathcal{A}$ is a *valid t-input adversary* if for all private-key t-input functional encryption schemes over message space $(\mathcal{X}_1)_\lambda \times \ldots \times (\mathcal{X}_t)_\lambda$, and a circuit space $\mathcal{F}$, for any (f_0, f_1) queried by the adversary, and any t pairs of input distribution ensembles $(D_{01}(\lambda), D_{11}(\lambda)), \ldots, (D_{0t}(\lambda), D_{1t}(\lambda))$[6] output by the adversary such that D_{bj} is a distribution over $\mathcal{X}_j$ for $b \in \{0, 1\}$, $j \in [t]$, it holds that

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

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

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

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

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

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

Definition 2.6.7. A t-input functional encryption scheme $t\text{-}\mathsf{CktFE}$ for a circuit family $\mathcal{F}$ is secure in the distributional indistinguishability game, if for all valid PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

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

2.6.3 Definitions: FE for Turing Machines

The definition of Turing machines and oblivious Turing machines was already recalled in Section 2.6.1. In this section, we will define functional encryption for Turing Machines (TM). Functional encryption for TMs is defined analogously to functional encryption for circuits, except that secret keys correspond to TMs rather than circuits. Thus, secret keys can be used to decrypt ciphertexts of messages of arbitrary length and the decryption time depends only the input-specific run time of the TM on the message, not the worst case run time. We denote the runtime of a TM M (i.e. number of steps the head takes) on an input $\mathbf{w}$ by $\mathsf{runtime}(M, \mathbf{w})$.

Single Input Functional Encryption for Turing Machines

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

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

Definition 2.6.8 (Correctness)**.** A functional encryption scheme TMFE is correct if for all $M \in \mathcal{M}$ and all $\mathbf{x} \in \Sigma^*$,

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

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

Efficiency [Ananth and Sahai (2016)]. The efficiency property of a public-key FE scheme for Turing machines says that the algorithm $\mathsf{TMFE.Setup}$ on input 1^λ should run in time polynomial in λ, $\mathsf{TMFE.KeyGen}$ on input the Turing machine M and the master key MSK should run in time polynomial in $(\lambda, |M|)$, $\mathsf{TMFE.Enc}$ on input a message $\mathbf{x}$ and the public key should run in time polynomial in $(\lambda, |\mathbf{x}|)$. Finally, $\mathsf{TMFE.Dec}$ on input a functional key of M and an encryption of $\mathbf{x}$ should run in time polynomial in $(\lambda, |M|, |\mathbf{x}|, \mathsf{runtime}(M, \mathbf{x}))$.

Distributional Indistinguishability for TMFE**.** In this section we define the notion of distributional indistinguishability based security for functional encryption for Turing machines. This notion was first considered by [Gentry *et al.* (2014)] in the context of single key FE for circuits.

Definition 2.6.9. A functional encryption scheme $\mathcal{F}$ for a TM family $\mathcal{M}$ is secure in the distributional indistinguishability game, if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter λ:

1. Public Key: Challenger returns PK to the adversary.
2. Pre-Challenge Key Queries: $\mathcal{A}$ may adaptively request keys for any TMs $M_i \in \mathcal{M}$. In response, $\mathcal{A}$ is given the corresponding keys SK_{M_i}. This step may be repeated any polynomial number of times by the attacker.
3. Challenge Declaration: $\mathcal{A}(1^\lambda, \mathsf{PK})$ outputs two challenge distribution ensembles $(D_0(\lambda), D_1(\lambda))$[7][8] to the challenger, subject to the restriction that for any $\mathbf{x}_0 \leftarrow D_0, \mathbf{x}_1 \leftarrow D_1$, it holds that $M_i(\mathbf{x}_0) \overset{c}{\approx} M_i(\mathbf{x}_1)$ for all i.
4. Challenge CT: $\mathcal{A}$ requests the challenge ciphertext, to which challenger chooses a random bit b, samples $\mathbf{x}_b \leftarrow D_b$ and returns the ciphertext $\mathsf{CT}_{\mathbf{x}_b}$.
5. Key Queries: The adversary may continue to request keys for additional functions, subject to the same restriction that for any $\mathbf{x}_0 \leftarrow D_0, \mathbf{x}_1 \leftarrow D_1$, it holds that $M_i(\mathbf{x}_0) \overset{c}{\approx} M_i(\mathbf{x}_1)$ for all i.
6. $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

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

Comparison with Standard Indistinguishability. We note that the standard indistinguishability game is implied by the above by restricting the adversary to choose distributions D_0, D_1 above to simply be two messages $\mathbf{x}_0, \mathbf{x}_1$ with probability 1 and requesting keys that satisfy $M_i(\mathbf{x}_0) = M_i(\mathbf{x}_1)$ for all i.

Multi-Input Functional Encryption for Turing Machines

In this section, we define multi input functional encryption (miFE) for Turing machines. Our definition generalizes the CktFE definitions of [Brakerski *et al.* (2016); Komargodski and Segev (2017)]. Our definition supports a fixed number of encryptors, where each of k (say) encryptors is associated with an index $\mathsf{ind} \in [k]$. An encryptor may choose an input string of unbounded length, denoted by ℓ_{ind}. We note that our definition is weaker than that of [Badrinarayanan *et al.* (2015)], who allow for unbounded number of

[7]We omit the argument λ where it is implicit for notational brevity.
[8]Describing these distributions by *efficiently computable* circuits is enough for our purposes.

encryptors and each encryptor to have a unique encryption key, a subset of which may be requested by the adversary. By contrast, our definition, following [Brakerski *et al.* (2016)], requires all encryptors to use the same MSK and evidently cannot allow the attacker to request this.

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

- $\mathsf{kTMFE.Setup}(1^\lambda, 1^k)$ is a PPT algorithm that takes as input the unary representation of the security parameter and the number of users k and outputs the master secret key MSK.
- $\mathsf{kTMFE.KeyGen}(\mathsf{MSK}, M)$ is a PPT algorithm that takes as input master secret key MSK and a TM M and outputs a corresponding secret key SK_M.
- $\mathsf{kTMFE.Enc}(\mathsf{MSK}, \mathbf{x}_{\mathsf{ind}}, \mathsf{ind})$ is a PPT algorithm that takes as input the master secret key MSK, an index $\mathsf{ind} \in [k]$ denoting the party number, and an input message $\mathbf{x}_{\mathsf{ind}}$ of arbitrary length and outputs a ciphertext $\mathsf{CT}_{\mathbf{w}_{\mathsf{ind}}}$.
- $\mathsf{kTMFE.Dec}(\mathsf{SK}_M, \{\mathsf{CT}_{\mathbf{x}_{\mathsf{ind}}}\}_{\mathsf{ind}\in[\mathsf{k}]})$ is an (a deterministic) algorithm that takes as input the functional secret key SK_M and k ciphertexts $\mathsf{CT}_{\mathbf{x}_1}, \dots, \mathsf{CT}_{\mathbf{x}_k}$ and outputs a bit b.

Definition 2.6.10 (Correctness)**.** A functional encryption scheme kTMFE is correct if for all $M \in \mathcal{M}$ and all $\mathbf{x}_i \in \Sigma^*$ for $i \in [k]$,

$$\Pr\left[\begin{array}{l} \mathsf{kTMFE.Dec}\Big(\mathsf{kTMFE.KeyGen}(\mathsf{MSK}, M), \mathsf{kTMFE.Enc}(\mathsf{MSK}, \mathbf{x}_1, 1), \\ \dots, \mathsf{kTMFE.Enc}(\mathsf{MSK}, \mathbf{x}_k, k)\Big) \neq \mathsf{M}(\mathbf{x}_1\|\dots\|\mathbf{x}_k) \end{array}\right] = \mathrm{negl}(\lambda)$$

where $\mathsf{MSK} \leftarrow \mathsf{kTMFE.Setup}(1^\lambda, 1^k)$ and the probability is taken over the coins of $\mathsf{kTMFE.Setup}$, $\mathsf{kTMFE.KeyGen}$, and $\mathsf{kTMFE.Enc}$.

Efficiency is as defined in Section 2.6.3, with runtimes to polynomially depend on k.

Distributional Indistinguishability for kTMFE. In this section we define the notion of distributional indistinguishability based security for multi-input functional encryption for Turing machines. To begin, we define the notion of a valid k-input adversary analogously to the case of circuits [Brakerski *et al.* (2016)].

Definition 2.6.11 (Valid k-Input Adversary)**.** A PPT algorithm $\mathcal{A}$ is a *valid k-input adversary*, if for a Turing machine space $\mathcal{M}$ with alphabet Σ, for all private key k-input functional encryption schemes kTMFE over message space $\mathcal{X}_1^* \times \ldots \times \mathcal{X}_k^*$ such that $\mathcal{X}_j^* \subset \Sigma^*$ for all $j \in [k]$, for any $M \in \mathcal{M}$ queried by the adversary, and any k pairs of input distribution ensembles $(D_{01}(\lambda), D_{11}(\lambda)), (D_{02}(\lambda), D_{12}(\lambda)), \ldots, (D_{0k}(\lambda), D_{1k}(\lambda))$[9][10] output by the adversary such that D_{bj} is a distribution over $\mathcal{X}_j^*$ for $b \in \{0,1\}$, $j \in [k]$, it holds that

$$M(\mathbf{x}_{01} \| \ldots \| \mathbf{x}_{0k}) \overset{c}{\approx} M(\mathbf{x}_{11} \| \ldots \| \mathbf{x}_{1k})$$

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

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

1. **Key Queries.** $\mathcal{A}$ may adaptively submit key requests for TMs $M_i \in \mathcal{M}$. In response, $\mathcal{A}$ is given the corresponding keys SK_{M_i} for some random bit b chosen by the challenger. This step may be repeated any polynomial number of times by the attacker.
2. **Ciphertext Queries.** $\mathcal{A}(1^\lambda)$ submits ciphertext requests for k pairs of challenge distribution ensembles $(D_{01}, D_{11}), (D_{02}, D_{12}), \ldots, (D_{0k}, D_{1k})$ to the challenger. The challenger samples $\mathbf{x}_{bj} \leftarrow D_{bj}$ for $j \in [k]$ and returns $\mathsf{kTMFE.Enc}(\mathsf{MSK}, \mathbf{x}_{bj}, j)$ for all $j \in [k]$. This step may be repeated any polynomial number of times by the attacker.
3. **Guess.** $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

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

Definition 2.6.12. A multi input functional encryption scheme kTMFE for a TM family $\mathcal{M}$ is secure in the distributional indistinguishability game, if for all valid PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

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

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

[10]Describing these distributions by *efficiently computable* circuits is enough for our purposes.

Indistinguishability Obfuscation for Turing Machines

All relevant and prior work [Bitansky *et al.* (2015*a*); Canetti *et al.* (2014); Koppula *et al.* (2015); Ananth *et al.* (2017)] constructs iO for Turing machines (TMs) in the setting where the input length is fixed a-priori. Essentially, the size of the obfuscated TM grows with input bound for these constructions. Further, [Lin *et al.* (2016)] showed that iO for TMs with *unbounded* inputs can be built starting from the notion of output-compressing randomized encodings in the plain model for TMs and that it is impossible to construct such randomized encodings, in general. However, this still does not rule out the existence of iO for unbounded-input TMs though. We construct iO for TMs but our techniques limit the construction also to work for bounded inputs only, as obtained in all prior work.

A uniform PPT machine iO is an indistinguishability obfuscator for a class of Turing machines $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ with input length L, if the following conditions are satisfied:

1. **Correctness.** For all security parameters $\lambda \in \mathbb{N}$, for any $M \in \mathcal{M}_\lambda$ and every input $\mathbf{x} \in \{0,1\}^{\leq L}$, we have that:

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

   where the probability is taken over the coin-tosses of the obfuscator iO.

2. **Indistinguishability of Equivalent TMs.** For every ensemble of pairs of Turing machines $\{M_{0,\lambda}, M_{1,\lambda}\}_{\lambda\in\mathbb{N}}$, such that $M_{0,\lambda}(\mathbf{x}) = M_{1,\lambda}(\mathbf{x})$ for every $\mathbf{x} \in \{0,1\}^{\leq L}$ and $\mathsf{runtime}(M_{0,\lambda}, \mathbf{x}) = \mathsf{runtime}(M_{1,\lambda}, \mathbf{x})$, we have that the following ensembles of pairs of distributions are indistinguishable to any PPT adversary A:

$$\left\{M_{0,\lambda}, M_{1,\lambda}, \mathsf{iO}(1^\lambda, M_{0,\lambda})\right\} \overset{c}{\approx} \left\{M_{0,\lambda}, M_{1,\lambda}, \mathsf{iO}(1^\lambda, M_{1,\lambda})\right\}$$

3. **Succinctness.** For all security parameters $\lambda \in \mathbb{N}$, for any $M \in \mathcal{M}_\lambda$, we have that the running time of $\mathsf{iO}(1^\lambda, M, L)$ is $\mathrm{poly}(\lambda, |M|, L)$ and the evaluation time of $\mathsf{iO}(M)$ on input $\mathbf{x}$ where $\mathbf{x} \in \{0,1\}^{\leq L}$, is $\mathrm{poly}(|M|, L, t)$ where $t = \mathsf{runtime}(M, \mathbf{x})$.

2.6.4 Constrained Pseudorandom Functions

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

Definition 2.6.13. [Boneh and Waters (2013)] Let $F : \{0,1\}^{\mathsf{seed}(\lambda)} \times \{0,1\}^{\mathsf{in}(\lambda)} \to \{0,1\}^{\mathsf{out}(\lambda)}$ be an efficient function, where seed, in and out are all polynomials in the security parameter λ. We say that F is a delegatable constrained pseudorandom function with respect to a set system $\mathcal{S} \subseteq 2^{\{0,1\}^{\mathsf{in}(\lambda)}}$ if there exist algorithms $(\mathsf{Setup}, \mathsf{Constrain}, \mathsf{Eval}, \mathsf{KeyDel})$ that satisfy the following:

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

Note that a set system is equivalent to a function family by defining set S as the set of inputs where the function evaluates to 1. For our purposes, it will be more convenient to represent sets as functions.

Security. Constrained security is defined using the following two experiments denoted $\mathsf{EXP}(0)$ and $\mathsf{EXP}(1)$ with an adversary $\mathcal{A}$. For $b \in \{0,1\}$ experiment $\mathsf{EXP}(b)$ proceeds as follows:

First, a random key $k \in \{0,1\}^{\mathsf{seed}(\lambda)}$ is selected and two helper sets $C, V \subseteq \{0,1\}^{\mathsf{in}}$ are initialized to $\varnothing$. The set V will keep track of all the points at which the adversary can evaluate. The set C will keep track of the points where the adversary has been challenged. The sets C and V will ensure that the adversary cannot trivially decide whether challenge values are random or pseudorandom. In particular, the experiments maintain the invariant that $C \cap V = \varnothing$.

The adversary $\mathcal{A}$ is presented with three oracles as follows:

1. $F.\mathsf{Eval}$: Given $\mathbf{x} \in \{0,1\}^{\mathsf{in}}$, if $\mathbf{x} \notin C$, the oracle returns $F(\mathsf{sk}, \mathbf{x})$, else it returns $\perp$. The point x is added to set V.
2. $F.\mathsf{Constrain}$: Given a set $S \in \mathcal{S}$ from $\mathcal{A}$, if $S \cap C = \varnothing$ the oracle returns $F.\mathsf{Constrain}(\mathsf{sk}, S)$, otherwise returns $\perp$. The set V is updated to contain S.

3. Challenge: Given $\mathbf{x}$ from $\mathcal{A}$, where $\mathbf{x} \notin V$, if $b = 0$, the adversary is given $F(\mathsf{sk}, \mathbf{x})$, else a random (consistent) element y. The set C is updated to contain $\mathbf{x}$.

When the adversary is done interrogating the oracles, it outputs a bit b'. Let W_b be the event that $b' = 1$ in $\mathsf{EXP}(b)$. The adversary's advantage is defined as $|\Pr[W_0] - \Pr[W_1]|$. We say that the PRF F is a secure constrained PRF with respect to a set system $\mathcal{S}$ if all PPT adversaries $\mathcal{A}$ have negligible advantage in the above game.

2.7 Construction: Single-Input FE for Turing Machines

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

1. Two compact functional encryption schemes for circuits, $\mathsf{1FE}_1$ and $\mathsf{1FE}_2$. We will assume that the scheme $\mathsf{1FE}_2$ is decomposable as per Definition 2.6.2.
2. A symmetric encryption scheme $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$.
3. A delegatable constrained pseudorandom function (cPRF), denoted by F which supports T delegations for the function family $f_t : \{0,1\}^{2\cdot\lambda} \to \{0,1\}$ defined as follows. Let x, t denote integers whose binary representations are $\mathbf{x}, \mathbf{t}$ of λ bits. Then,

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

Intuitively, the function is parametrized by a value t and evaluates to 1 if the first half of its input satisfies $x \geq t$. We will denote the constrained PRF key K_{f_t} corresponding to function f_t by K_t for ease of notation. By the delegation property of constrained PRFs (Section 2.6.4), we have that if $t' \geq t$ then $\mathsf{K}_{t'}$ can be derived from K_t. In our construction the parameter t will represent the time step in the computation, which means that a PRF key of the current time step can be used to derive PRF keys for future time steps. We will denote a PRF for this functionality by F. The security proof makes use of a punctured version of the above cPRF, please see Sections 2.7.3 and A.3 for details.

2.7.1 Construction of Single-Input TMFE

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

Notation. Note that since $\mathsf{1FE}_2$ is decomposable, there exists an encoding function $\mathcal{E}$ which encodes each bit of the input and since it is compact, the output length of $\mathcal{E}$

is independent of the circuit class supported by $1\mathsf{FE}_2$. Thus, by choosing the encoding function first, the CktFE scheme may support a circuit class that outputs its own ciphertext components. We denote by $\bar{\mathcal{E}}$ the encoding function $\mathcal{E}$ applied bitwise to a vector, i.e. $\bar{\mathcal{E}}(\mathbf{w}) = \mathcal{E}(w_1)\ldots\mathcal{E}(w_n)$.

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

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

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

   Here, Σ and $\mathcal{Q}$ are the alphabet and state space respectively of the Turing machine family. The tokens SYM and ST are flags denoting a symbol and a state respectively. The set $\{0,1\}^{4\lambda}$ encodes in order, a random value key-id associated with a TM M, a cPRF key, the current time step in the computation and the length of the input string, each of λ bits. Here, Trap is a data structure of fixed polynomial length which will be used in the proof. Since we do not need it in the construction, we do not discuss it here, please see Figure 2.6 for its definition. $\mathcal{C}^{1\mathsf{FE}_2}$ denotes the ciphertext space of $1\mathsf{FE}_2$, and ACC and REJ are bits indicating accepting and a rejecting states of a TM respectively.

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

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

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

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

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

1. Sample the root key K_0 for function f_t where $t = 0$ for the cPRF F described above.
2. For $i \in [\ell]$, let $\mathsf{CT}_i = 1\mathsf{FE}_1.\mathsf{Enc}(\mathsf{PK}, (\mathsf{K}_0, i, \ell, w_i, \mathsf{Trap}))$, where Trap is a data structure which is only relevant in the proof. Here, all fields of Trap are set to $\perp$ except a flag $\mathsf{Trap.mode\text{-}real} = 1$ which indicates that we are in the real world. Please see Figure 2.6 for the definition of Trap.
3. Output $\mathsf{CT}_{\mathbf{w}} = \{\mathsf{CT}_i\}_{i\in[\ell]}$.

$\mathsf{TMFE.KeyGen}(\mathsf{MSK}, M)$: Upon input the master secret key MSK and the description of a Turing machine M, do the following. We will assume, w.l.o.g. that the TM is oblivious (see Appendix 2.6.1 for a justification) and $\mathsf{q}_{\mathsf{st}} \in \mathcal{Q}$ is the start state of M.

1. Sample a random value $\mathsf{salt} \leftarrow \{0,1\}^\lambda$.
2. Interpret $\mathsf{MSK} = (\mathsf{1FE}_1.\mathsf{MSK}, (\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}))$.
3. Let $\mathsf{SK}_{\mathsf{ReRand}} = \mathsf{1FE}_1.\mathsf{KeyGen}(\mathsf{1FE}_1.\mathsf{MSK}, \mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp})$ where Figure 2.2 defines the circuit $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$.

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

(a) **Initialization and Choosing Real or Trapdoor mode.**
Initialize an input vector $\mathsf{inp} = (w_i, \mathsf{q}_{\mathsf{st}})$. If $\mathsf{Trap.mode\text{-}real} = 1$, set $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. If $i \neq 1$, set $\mathsf{inp} = (w_i, \perp)$. Else invoke $\mathsf{Trap\text{-}Mode}_{\mathsf{ReRand}}(\mathsf{Trap}, \mathsf{inp}, \mathsf{salt}, \ell, \mathcal{C}_1, \mathcal{C}_2, i)$ as described in Figure 2.3 to obtain $\big(\mathsf{inp} = (u_1, u_2), \mathsf{out} = (c_1, c_2)\big)$.

(b) **Computing Encrypted Symbols using randomness derived from** cPRF**.** If $\mathsf{out}.c_1 = \perp$, do the following.
   i. Noting that $i > 0$, derive delegated cPRF key K_i from K_0 as $\mathsf{K}_i = \mathsf{F.KeyDel}(\mathsf{K}_0, f_i)$. Compute randomness for encryption as $r_i = \mathsf{F.Eval}(\mathsf{K}_i, (i \| \mathsf{salt}))$.
   ii. Derive delegated cPRF key $\mathsf{K}_{i+1} = \mathsf{F.KeyDel}(\mathsf{K}_i, f_{i+1})$. Set $\mathsf{key\text{-}id} = \mathsf{salt}$.
   iii. Compute the $\mathsf{1FE}_2$ ciphertext component encoding $w_i = \mathsf{inp}.u_1$ for time step i as
$$\mathsf{CT}_{\mathsf{sym},i} = \bar{\mathcal{E}}\big(\mathsf{1FE}_2.\mathsf{PK}_1, (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{i+1}, i, \ell, w_i, \mathsf{Trap}); r_i\big)$$
   iv. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},i}$.

(c) **Computing Encrypted State for First Time Step.** If $\big((\mathsf{out}.c_2 = \perp) \wedge (i = 1)\big)$, do the following.
   i. Compute $\mathsf{1FE}_2$ ciphertext component to encode the starting state $\mathsf{q}_{\mathsf{st}} = \mathsf{inp}.u_2$ as
$$\mathsf{CT}_{\mathsf{st},1} = \bar{\mathcal{E}}\big(\mathsf{1FE}_2.\mathsf{PK}_2, (\mathsf{ST}, \mathsf{q}_{\mathsf{st}}); r_1\big)$$
   ii. Set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},1}$.

(d) **Output :** If $i = 1$, output $\mathsf{out} = (c_1, c_2)$, else output $\mathsf{out} = (c_1, \perp)$.

Figure 2.2: This circuit re-randomizes the ciphertexts provided during encryption to use randomness derived from a cPRF. The seed for the cPRF is specified in the ciphertext and the input is specified by the key. This ensures that each ciphertext, key pair form a unique "thread" of execution.

4. Let $\mathsf{SK}_{\mathsf{Next}} = \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\perp,\perp})$ where Figure 2.4 defines the circuit $\mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\perp,\perp}$.
5. Output $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$.

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

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

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

(a) If $\mathsf{Trap.mode\text{-}trap}_3 = 1$, do the following:

- i. If $\big((\mathsf{Trap.Sym\ \ TS} = i) \wedge (i \leq \ell)\big)$, compute the $1\mathsf{FE}_2$ ciphertext $\mathsf{CT}_{\mathsf{sym},i} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_1)$ and set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},i}$.
- ii. If $\big((\mathsf{Trap.ST\ \ TS} = i) \wedge (i = 1)\big)$, compute the $1\mathsf{FE}_2$ ciphertext $\mathsf{CT}_{\mathsf{st},i} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},1}$.

(b) If $\mathsf{Trap.mode\text{-}trap}_1 = 1$, do the following:

- i. If $\big((\mathsf{Trap.Sym\ TS}_1 = i) \wedge (i \leq \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol to be encrypted and output at time step i.
- ii. If $\big((\mathsf{Trap.ST\ TS}_1 = i) \wedge (i = 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_1$ with the start state to be encrypted and output at time step 1.

(c) If $\mathsf{Trap.mode\text{-}trap}_2 = 1$, do the following:

- i. If $\big((\mathsf{Trap.Sym\ TS}_2 = i) \wedge (i \leq \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol to be encrypted and output at time step i.
- ii. If $\big((\mathsf{Trap.ST\ TS}_2 = i) \wedge (i = 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_2$ with the start state to be encrypted and output at time step 1.

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

(a) If $\mathsf{salt} > \mathsf{Trap.key\text{-}id}$ set $b = 0$; else set $b = 1$[a].

(b) If $i \neq 1$, update $\mathsf{inp} = (\mathsf{Trap.val}_b, \perp)$; else update $\mathsf{inp} = (\mathsf{Trap.val}_b, \mathsf{q_{st}})$.

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

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

Figure 2.3: Subroutine handling the trapdoor modes in ReRand. This is "active" only in the proof.

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

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

Function $\mathsf{Next}_{\mathsf{1FE_2.PK},\mathsf{salt},M,\mathcal{C}_1,\mathcal{C}_2}\big((\mathbf{z}_1,\mathbf{z}_2)\big)$

(a) **Reading Current (Symbol, State) Pair and Looking up Transition Table.**

i. Interpret $\mathbf{z}_1 = (\mathsf{type}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, s, \mathsf{Trap})$, $\mathbf{z}_2 = (\mathsf{type}, s)$. If $((\mathbf{z}_1.\mathsf{type} \neq \mathsf{SYM}) \vee (\mathbf{z}_2.\mathsf{type} \neq \mathsf{ST}) \vee (\mathbf{z}_1.\mathsf{key\text{-}id} \neq \mathsf{salt}))$, output $\perp$ and abort.

ii. Interpret $(\mathbf{z}_1.s, \mathbf{z}_2.s) = (\sigma_t, \mathsf{q}_t)$ as the symbol, state pair for the current time step $t = \mathbf{z}_1.t$, input $\mathsf{K}_{t+1} = \mathbf{z}_1.\mathsf{K}_{t+1}$ as the constrained PRF key for future time steps. Denote $\mathsf{key\text{-}id} = \mathbf{z}_1.\mathsf{key\text{-}id}$, $\ell = \mathbf{z}_1.\ell$ and $\mathsf{Trap} = \mathbf{z}_1.\mathsf{Trap}$. Using the transition table of the machine M, look up the next state q_{t+1} as well as the symbol $\sigma_{t'}$ to be written on the work-tape, where t' is the time step the current work tape cell will next be read by M. If q_{t+1} is an accept or reject state, then output ACC or REJ and exit.

iii. Initialize $\mathsf{inp} = (\sigma_{t'}, \mathsf{q}_{t+1})$.

(b) **Choosing Real or Trapdoor mode.** If $\mathsf{Trap.mode\text{-}real} = 1$, initialize an output vector $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. Else invoke $\mathsf{Trap\text{-}Mode}_{\mathsf{Next}}\big(\mathsf{Trap}, \mathsf{inp}, \mathsf{salt}, \ell, \mathcal{C}_1, \mathcal{C}_2, t, t'\big)$ as described in Figure 2.5 to obtain $\big(\mathsf{inp}{=}(u_1,u_2), \mathsf{out}{=}(c_1,c_2)\big)$.

(c) **Computing Next Encrypted Symbol.** If $\mathsf{out}.c_1 = \perp$, do the following.

i. Noting that $t' > t$, derive the randomness at time step t' using the delegated key K_{t+1} as $r_{t'} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t' \| \mathsf{salt}))$. Compute the delegated PRF key $\mathsf{K}_{t'+1} = \mathsf{F.KeyDel}(\mathsf{K}_{t+1}, f_{t'+1})$.

ii. Compute the $\mathsf{1FE_2}$ ciphertext component encoding the symbol $\sigma_{t'} = \mathsf{inp}.u_1$ for time step t' as

$$\mathsf{CT}_{\mathsf{sym},t'} = \bar{\mathcal{E}}\big(\mathsf{1FE_2.PK_1}, (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t'+1}, t', \ell, \sigma_{t'}, \mathsf{Trap}); r_{t'}\big)$$

iii. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.

(d) **Computing Next Encrypted State.** If $\mathsf{out}.c_2 = \perp$, do the following.

i. Derive the randomness at time step $t+1$ as $r_{t+1} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t+1 \| \mathsf{salt}))$ and compute the $\mathsf{1FE_2}$ ciphertext component encoding the state $\mathsf{q}_{t+1} = \mathsf{inp}.u_2$ for time step $t+1$ as

$$\mathsf{CT}_{\mathsf{st},t+1} = \bar{\mathcal{E}}\big(\mathsf{1FE_2.PK_2}, (\mathsf{ST}, \mathsf{q}_{t+1}); r_{t+1}\big)$$

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

(e) **Output :** $\mathsf{out} = (c_1, c_2)$.

Figure 2.4: Function to mimic TM computation. It reads the current symbol, state pair and outputs an encryption of the new state and symbol to be written under the appropriate randomness generated using a cPRF.

- $\big(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1}\big)$.

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

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

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

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

(a) If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.mode\text{-}trap}_3 = 1)\big)$, do the following.

i. If $\big((\mathsf{Trap.Sym\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t') \wedge (t > \ell)\big)$, compute the $1\mathsf{FE}_2$ symbol ciphertext $\mathsf{CT}_{\mathsf{sym},t'} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_1)$ and set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.

ii. If $\big((\mathsf{Trap.ST\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t+1) \wedge (t > 1)\big)$, compute the $1\mathsf{FE}_2$ state ciphertext $\mathsf{CT}_{\mathsf{st},t+1} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},t+1}$.

(b) If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.mode\text{-}trap}_1 = 1)\big)$, do the following:

i. If $\big((\mathsf{Trap.Sym\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_1$ to be encrypted and given as output for time step t'.

ii. If $\big((\mathsf{Trap.ST\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_1$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_1$ to be encrypted and given as output for time step $t+1$.

(c) If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.mode\text{-}trap}_2 = 1)\big)$, do the following:

i. If $\big((\mathsf{Trap.Sym\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_2$ to be encrypted and given as output for time step t'.

ii. If $\big((\mathsf{Trap.ST\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_2$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_2$ to be encrypted and given as output for time step $t+1$.

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

Figure 2.5: Subroutine handling the trapdoor modes in Next. This is "active" only in the proof.

2.7.2 Correctness and Efficiency of Single-Input TMFE

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

1. Since CT_1 encodes Trap with $\mathsf{Trap.mode\text{-}real} = 1$, hence by the correctness of the $1\mathsf{FE}_1$ scheme, we get $1\mathsf{FE}_1.\mathsf{Dec}(\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{CT}_1) = (\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$ as output.

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

The new sequence of $\mathsf{1FE}_2$ ciphertexts output by ReRand are now sequenced as $\big((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \mathsf{CT}_{\mathsf{sym},2}, \ldots, \mathsf{CT}_{\mathsf{sym},|\mathbf{w}|}\big)$. The $\mathsf{1FE}_2$ scheme supports the functionality $\mathsf{Next} := \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathcal{C}_1,\mathcal{C}_2}$. Throughout the $\mathsf{1FE}_2$ decryption, we maintain the invariant that at any time step t, apart from a secret key $\mathsf{SK}_{\mathsf{Next}}$, the input to the $\mathsf{1FE}_2.\mathsf{Dec}$ algorithm is an entire $\mathsf{1FE}_2$ ciphertext decomposed into two components corresponding to a symbol and a state ciphertext both of which are computed with the same randomness, which is computed as $\mathsf{F.Eval}(\mathsf{K}_0, (\mathsf{t}\|\mathsf{salt}))$[11].

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

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

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

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

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

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

2.7.3 Proof of Security for Single-Input TMFE

Next, we prove that the above TMFE scheme satisfies distributional indistinguishability (DI) for single (or constant) length outputs, as long as the underlying CktFE scheme satisfies distributional indistinguishability for any output length. In Appendix A.4, we provide an instantiation of a CktFE scheme satisfying distributional indistinguishability.

Theorem 2.7.1. *Assume that the functional encryption schemes for circuits* $\mathsf{1FE}_1$ *and* $\mathsf{1FE}_2$ *are* DI *secure (according to definition 2.6.4) and that* F *is a secure* cPRF *for the function family defined above (according to definition 2.6.13). Then, the construction of functional encryption for Turing machines* TMFE *is selective* DI *secure for single bit outputs (according to definition 2.6.9).*

Since the intuition was discussed in Section 2.4, we proceed to the formal proof.

The Trapdoor Data Structure. To implement the approach discussed in Section 2.4, we will make use of a data-structure Trap that lets us store all the requisite trapdoor information needed for the security proof within the ciphertext. In our construction, decryption of a particular input by a particular function key results in a chain of ciphertexts, each of which contain the trapdoor data structure. In the real world, this information is not used but as we progress through the proof, different fields become relevant. The data structure is outlined in Figure 2.6.

Row 1. Above, key-id refers to the particular function key being considered and we switch the execution chain from $b = 0$ to $b = 1$ key by key. All the ciphertexts in

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

Figure 2.6: Data Structure Trap used for Proof

a given execution chain share the key-id value. We assume a lexicographic order on the key-id fields, this can be easily ensured by having a counter as part of the key-id field. We do not make this explicit below for notational brevity. If key-id* is the key identity programmed in a particular execution chain, then all keys with values smaller than key-id* will decrypt the chain using the input bit $b = 1$, and all keys with values larger than key-id* will use $b = 0$. Hence, the $1\mathsf{FE}_1$ ciphertexts provided by the encryptor must encode messages corresponding to both values of b, the fields val_0 and val_1 are designed for this purpose[13]. Note that $1\mathsf{FE}_2$ ciphertexts computed by decryption need not track messages corresponding to both values of b, since the "chain is extended" via decryption corresponding to exactly one of $b = 0$ or $b = 1$ depending on the relation between the key identities in the ciphertext and the function key. The field SKE.K refers to the key of a symmetric key encryption scheme, which is used to decrypt some encrypted value embedded in the function key. This is a standard trick when the key must hide something in the public key setting. The flag mode-real means the scheme operates in the real world mode and the trapdoor information is not used.

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

[13]For the knowledgeable reader, this is similar to what was done by Ananth and Jain (2015).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Indistinguishability follows from security of cPRF as discussed above.

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

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

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

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

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

Indistinguishability follows from security of SKE as above.

$\mathcal{H}(2,3)$: Set $\mathsf{mode\text{-}trap}_1 = 1$ with target $T-1$, $\mathsf{mode\text{-}trap}_2 = \perp$, and $\mathsf{mode\text{-}trap}_3 = 1$ with target $T-2$.

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

$\mathcal{H}(2,4)$: Switch normal root key to punctured key at point $(T-2\|\mathsf{key\text{-}id})$.

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

$\mathcal{H}(2,5)$: Switch randomness to true in the ciphertext hardwired in key.

Indistinguishability follows from security of cPRF as discussed above.

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

Indistinguishability follows from security of $\mathsf{1FE}_2$.

$\mathcal{H}(2,7)$: Switch randomness back to use the PRF in the ciphertext hardwired in key.

Indistinguishability follows from security of cPRF as discussed above.

$\mathcal{H}(2,8)$: Switch punctured root key to normal root key.

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

$\mathcal{H}(3,1)$: Intuitively, we slide the trapdoor left by one step, i.e. change target time-steps to $T-2$ and $T-3$ in the ciphertext. Now slot 1 for $T-2$ corresponds to $b=1$ and slot 2 for $T-3$ to $b=0$. Set $\mathsf{mode\text{-}real} = \mathsf{mode\text{-}trap}_3 = \perp$ and $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = 1$.

Indistinguishability follows from security of $\mathsf{1FE}_1$, since the decryption values in both hybrids are exactly the same. Note that now slot $T-1$ is redundant, since $T-2$ ciphertext is already switched to $b=1$.

Hybrid $\mathcal{H}(3,i)$ will be analogous to $\mathcal{H}(2,i)$ for $i \in [8]$.

As we proceed left in the execution chain one step at a time, we reach step ℓ where $\ell = |\mathbf{w}|$, i.e. time steps for which $\mathsf{1FE}_1$ ciphertexts are provided by the encryptor. At this point we will hardwire the ReRand key with symbol ciphertexts for ℓ time steps, one at a time, and the Next key for the state ciphertexts[14]. Moreover, we must now add an additional hybrid in which the challenge $\mathsf{1FE}_1$ ciphertext at position ℓ contains the message bit corresponding to $b=1$; intuitively, we must switch the bit before we slide the trapdoor since the ciphertext for this position is not generated by decrypting the previous ciphertext. In more detail, in $\mathcal{H}(T-\ell,8)$, analogously to hybrid $(1,8)$, the $T-(T-\ell) = \ell^{th}$ bit hard-wired in the trapdoor is changed to 1. We now add one more hybrid, namely:

$\mathcal{H}(T-\ell,9)$: In this hybrid, we modify the $\mathsf{1FE}_1$ challenge ciphertext in position ℓ as follows: the encoded message is changed corresponding to $b=1$ and flag $\mathsf{mode\text{-}real} = 1$. The other flags $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = \mathsf{mode\text{-}trap}_3 = \perp$.

Note that all ciphertexts previous to time step ℓ remain unchanged, and output their corresponding symbol ciphertexts correctly. The Next circuit outputs the state ciphertext for time step ℓ corresponding to bit $b=1$. The only difference between this hybrid and the previous one is that here we use the real mode to output the symbol ciphertext for $b=1$ whereas previously we used the trapdoor mode to output the same symbol ciphertext. Hence, decryption values in both hybrids are exactly the same, and indistinguishability follows from security of $\mathsf{1FE}_1$.

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

Finally in $\mathcal{H}(T-1,9)$, the entire chain has been replaced to use $b=1$ and all the challenge $\mathsf{1FE}_1$ ciphertexts have encoded messages corresponding to $b=1$ with $\mathsf{mode\text{-}real}=1$.

$\mathcal{H}(T)$: In this hybrid, all the other fields in the trapdoor data structure, excepting $\mathsf{mode\text{-}real}$ are disabled and set to $\perp$. This is the real world with $b=1$.

Since all the encoded messages use $b=1$, decryption values are all exactly the same as in $\mathcal{H}(T-1,9)$, hence indistinguishability follows from security of $\mathsf{1FE}_1$.

The formal reductions are provided in Appendix A.1.

Multiple Keys. We handle multiple keys by repeating the above set of hybrids key by key. Each key carries within it an identifier $\mathsf{key\text{-}id}$, and if this is less than the key identifier encoded in the ciphertext, the bit $b=1$ is used, if it is greater then the bit $b=0$ is used and if it is equal, then the above sequence of hybrids is performed to switch from $b=0$ to $b=1$. To support this, the $\mathsf{1FE}_1$ ciphertexts provided by the encryptor must encode messages corresponding to both values of b, the fields val_0 and val_1 in the trapdoor data structure of Figure 2.6 are provided for this purpose. Security follows by a standard hybrid argument as in [Ananth and Jain (2015)].

2.7.4 Constructing the cPRF.

In Appendix A.3, we provide a construction for a cPRF F which supports puncturing and delegation as required; the T cPRFs F_i for $i \in [T]$ may each be constructed similarly. To begin, note that we require the root key of F to be punctured at a point i^* (say). The cPRF construction for punctured PRF [Boneh and Waters (2013); Kiayias *et al.* (2013); Boyle *et al.* (2014)] (which is in turn inherited from the standard PRG based GGM [Goldreich *et al.* (1986)]) immediately satisfies this constraint, so we are left with the question of delegation.

Recall that we are required to delegate T times, where T is the (polynomial) runtime of the Turing machine on the encrypted input (please see Section 2.7), and the j^{th} delegated key must support evaluation of points $\{(k\|z) : z \in \{0,1\}^\lambda\}$ for $k \geq j$, *except* when $(k\|z) = i^*$. This may be viewed as the j^{th} key being punctured on points

$[1, j-1] \cup i^*$. We show that the GGM based construction for puncturing a single point can be extended to puncturing an interval (plus an extra point). Intuitively, puncturing an interval corresponds to puncturing at most λ internal nodes in the GGM tree. In more detail, we show that regardless of the value of j, it suffices to puncture at most λ points in the GGM tree to achieve puncturing of the entire interval $[1, j-1]$. Please see Appendix A.3 for details.

2.8 Construction: Multi-Input FE for Turing Machines

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

1. A compact, k-input functional encryption scheme for circuits, kFE and a compact, public-key functional encryption scheme 1FE. As before, we will assume that the scheme 1FE is decomposable as defined in Section 2.6.
2. A symmetric encryption scheme $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$.
3. A delegatable constrained pseudorandom function (cPRF), denoted by F which supports T delegations for the function family $f_t : \{0,1\}^{(k+2)\cdot\lambda} \to \{0,1\}$ defined as follows. Let x, t denote integers whose binary representations are $\mathbf{x}, \mathbf{t}$ of λ bits. Then,

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

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

Our k-input CktFE scheme may be either private or public key, and will result in the corresponding notion for k-input TMFE. Since the multi input setting for FE is considered more interesting in the symmetric key setting (see [Brakerski *et al.* (2016)] for a discussion), we present our construction in the symmetric key setting – the public key adaptation is straightforward.

We note that ciphertexts output by Agg, which are encryptions of the symbols in the aggregate string under the $\mathsf{1FE}$ scheme, are exactly the same as the output of the ReRand function in the single input scheme of Section 2.7. Therefore, as before, we may have the functionality Next of the $\mathsf{1FE}$ scheme mimic the computation of the Turing machine on the global string $(\mathbf{w}_1\|\ldots\|\mathbf{w}_k)$. As in the previous construction, $\mathsf{1FE.Dec}$ accepts as its inputs a ciphertext decomposed into two components encoding the current symbol on the worktape and the current state in the computation, both of which have been encrypted using the same randomness, and outputs a ciphertext component corresponding to the symbol written on the tape, as well as the next state. The global salt in the ciphertext, along with a random nonce chosen by KeyGen are used as input to a cPRF as before, to compute the randomness used to generate ciphertexts. This ensures that the execution of a given machine on a given input combination is maintained separate from any other execution, and thwarts "mix and match" attacks, where, for instance, an attacker may try to combine a state generated at some time step t in one execution with a symbol generated at time step t from a different execution.

If we instantiate the underlying multi-input CktFE by the construction of [Komargodski and Segev (2017)], we may let the arity k be poly-logarithmic in the security parameter. If we instantiate multi-input CktFE by the construction of [Goldwasser *et al.* (2014)], we may support fixed polynomial arity at the cost of worsening the assumption. Note that [Goldwasser *et al.* (2014)] rely on iO while [Komargodski and Segev (2017)] rely on compact FE. Note that [Badrinarayanan *et al.* (2015)] support unbounded polynomial arity, but from public coin DiO as discussed in Section 2.1.

2.8.1 Construction of Multi-Input TMFE

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

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

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

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

   The tokens SYM and ST are flags denoting a symbol and a state respectively of a Turing machine M which has Σ and $\mathcal{Q}$ as the alphabet and state space respectively. The set $\{0,1\}^{(k+4)\lambda}$ encodes in order, a random value $\mathsf{key\text{-}id}$ associated with a TM M, a constrained PRF key, the current time step in the computation, the length of the input string, each of λ bits and a string of length $k\cdot\lambda$ bits encoding a random value gsalt. Here, Trap is a data structure of fixed polynomial length which will be used in the proof. Since we do not need it in the construction, we do not discuss it here, please see Figure A.7 for its definition. The set $\{0,1\}^{k\cdot\lambda}$ encodes again a random value gsalt associated with the message component for state. $\mathcal{C}^{\mathsf{1FE}}$ is the ciphertext space of $\mathsf{1FE}$. ACC and REJ denote tokens when M reaches an accepting state and a rejecting state respectively.

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

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

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

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

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

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

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

1. Interpret the input $\mathsf{MSK} = (\mathsf{kFE.MSK}, (\mathsf{1FE.PK}, \mathsf{1FE.MSK}))$.
2. Let $\mathbf{w}_{\mathsf{ind}} = w_1 w_2 \ldots w_{\ell_{\mathsf{ind}}}$. Sample $\mathsf{salt}_{\mathsf{ind}} \leftarrow \{0,1\}^\lambda$.
3. Construct the data structure Trap and set all its fields to $\perp$ except a flag $\mathsf{Trap.mode\text{-}real} = 1$ which indicates that we are in the real world. The data structure Trap is only relevant in the proof. Please see Figure 2.6 for the definition of Trap.

- **Encoding Input String and Its Length**

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

$\mathsf{kTMFE.KeyGen}(\mathsf{MSK}, M)$: Upon input the master secret key MSK and the description of a Turing machine M, do the following. We will assume, w.l.o.g. that the TM is oblivious (see Appendix 2.6.1 for a justification) and $\mathsf{q}_{\mathsf{st}} \in \mathcal{Q}$ is the start state of M.

1. Sample a random value $\mathsf{rand} \leftarrow \{0,1\}^\lambda$.
2. Interpret $\mathsf{MSK} = (\mathsf{kFE.MSK}, (\mathsf{1FE.PK}, \mathsf{1FE.MSK}))$.
3. Let $\mathsf{SK}_{\mathsf{Agg}} = \mathsf{kFE.KeyGen}(\mathsf{kFE.MSK}, \mathsf{Agg}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{q}_{\mathsf{st}},\perp,\perp})$, where Figure 2.7 defines the circuit $\mathsf{Agg}_{\mathsf{1FE.PK},\mathsf{rand},\mathsf{q}_{\mathsf{st}},\perp,\perp}$.
4. Let $\mathsf{SK}_{\mathsf{Next}} = \mathsf{1FE.KeyGen}(\mathsf{1FE.MSK}, \mathsf{Next}_{\mathsf{1FE.PK},\mathsf{rand},M,\perp,\perp})$, where Figure 2.9 defines the circuit $\mathsf{Next}_{\mathsf{1FE.PK},\mathsf{rand},M,\perp,\perp}$.
5. Output the secret key as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{Agg}}, \mathsf{SK}_{\mathsf{Next}})$.

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

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

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

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

(c) **Initialization and Choosing Real or Trapdoor mode.**

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

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

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

iii. Else obtain $\big(\mathsf{inp} = (u_1,u_2), \mathsf{out} = (c_1,c_2)\big) = \mathsf{Trap\text{-}Mode}_{\mathsf{Agg}}\big(\mathsf{Trap},\mathsf{inp},\mathsf{rand},\mathsf{gsalt},\ell,\mathcal{C}_1,\mathcal{C}_2,\mathsf{pos}\big)$ as described in Figure 2.8.

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

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

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

iii. **Computing Encrypted Symbols using randomness derived from** cPRF**.** If $\mathsf{out}.c_1 = \perp$, do the following.

- Compute the delegated PRF key $\mathsf{K}_{\mathsf{pos}+1} = \mathsf{F.KeyDel}(\mathsf{K}_0, f_{\mathsf{pos}+1})$. Set $\mathsf{key\text{-}id} = \mathsf{rand}$.

- Compute the 1FE symbol ciphertext encoding $\sigma = \mathsf{inp}.u_1$ as $\mathsf{CT}_{\mathsf{sym},\mathsf{pos}} = \bar{\mathcal{E}}\,(\mathsf{1FE.PK}_1, \mathbf{y}_1; r_{\mathsf{pos}})$, where $\mathbf{y}_1 = (\mathsf{SYM},\mathsf{key\text{-}id},\mathsf{K}_{\mathsf{pos}+1},\mathsf{pos},\ell,\mathsf{gsalt},\sigma,\mathsf{Trap})$. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},\mathsf{pos}}$.

iv. **Computing Encrypted State for First Time Step.** If $((\mathsf{out}.c_2 = \perp) \wedge (\mathsf{pos} = 1))$, do the following.

- Compute the 1FE state ciphertext encoding $\mathsf{q_{st}} = \mathsf{inp}.u_2$ as $\mathsf{CT}_{\mathsf{st},1} = \bar{\mathcal{E}}(\mathsf{1FE.PK}_2, \mathbf{y}_2; r_1)$, where $\mathbf{y}_2 = (\mathsf{ST}, \mathsf{q_{st}}, \mathsf{gsalt})$. Set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},1}$.

(e) If $\mathsf{pos} = 1$, output $\mathsf{out} = (c_1, c_2)$. Otherwise, output $\mathsf{out} = (c_1, \perp)$.

Figure 2.7: This circuit aggregates and re-randomizes the ciphertexts provided during encryption to use randomness derived from a cPRF. The seed for the cPRF is specified in the ciphertext for first party and the input is specified by the key. This ensures that each ciphertext, key pair form a unique “thread” of execution.

```
Subroutine Trap-Mode_Agg(Trap, inp, rand, gsalt, ℓ, C_1, C_2, pos)

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

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

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

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

Figure 2.8: Subroutine handling the trapdoor modes in Agg. This is “active" only in the proof.

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

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

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

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

(a) **Reading Current (Symbol, State) Pair and Looking up Transition Table.**

i. Interpret $\mathbf{z}_1 = (\mathsf{type}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, \mathsf{gsalt}, s, \mathsf{Trap}), \mathbf{z}_2 = (\mathsf{type}, s, \mathsf{gsalt})$. If $((\mathbf{z}_1.\mathsf{type} \neq \mathsf{SYM}) \vee (\mathbf{z}_2.\mathsf{type} \neq \mathsf{ST}) \vee (\mathbf{z}_1.\mathsf{key\text{-}id} \neq \mathsf{rand}) \vee \wedge(\mathbf{z}_1.\mathsf{gsalt} \neq \mathbf{z}_2.\mathsf{gsalt}))$, output $\perp$ and abort.

ii. Interpret $(\mathbf{z}_1.s, \mathbf{z}_2.s) = (\sigma_t, q_t)$ as the symbol, state pair for the current time step $\mathbf{z}_1.t = t$, input $\mathbf{z}_1.\mathsf{K}_{t+1} = \mathsf{K}_{t+1}$ as the constrained PRF key for future time steps. Denote $\mathsf{key\text{-}id} = \mathbf{z}_1.\mathsf{key\text{-}id}$, $\ell = \mathbf{z}_1.\ell$, $\mathsf{gsalt} = \mathbf{z}_1.\mathsf{gsalt}$ and $\mathsf{Trap} = \mathbf{z}_1.\mathsf{Trap}$. Using the transition table of the machine M, look up the next state q_{t+1} as well as the symbol $\sigma_{t'}$ to be written on the work-tape, where t' is the time step the current work tape cell will next be read by M. If q_{t+1} is an accept or reject state, then output ACC or REJ and exit.

iii. Initialize $\mathsf{inp} = (\sigma_{t'}, \mathsf{q}_{t+1})$.

(b) **Choosing Real or Trapdoor mode.** If $\mathsf{Trap.mode\text{-}real} = 1$, initialize an output vector $\mathsf{out} = (c_1, c_2)$, where $c_1 = c_2 = \perp$. Else invoke $\mathsf{Trap\text{-}Mode}_{\mathsf{Next}}\big(\mathsf{Trap}, \mathsf{inp}, \mathsf{rand}, \mathsf{gsalt}, \ell, \mathcal{C}_1, \mathcal{C}_2, t, t'\big)$ as described in Figure 2.10 to obtain $\big(\mathsf{inp}{=}(u_1, u_2), \mathsf{out}{=}(c_1, c_2)\big)$.

(c) **Computing Next Encrypted Symbol.** If $\mathsf{out}.c_1 = \perp$, do the following.

i. Noting that $t' > t$, derive the randomness at time step t' using the delegated key K_{t+1} as $r_{t'} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t' \| \mathsf{rand} \| \mathsf{gsalt}))$. Compute the delegated PRF key $\mathsf{K}_{t'+1} = \mathsf{F.KeyDel}(\mathsf{K}_{t+1}, f_{t'+1})$.

ii. Compute the 1FE ciphertext component encoding the symbol $\sigma_{t'} = \mathsf{inp}.u_1$ for time step t' as

$$\mathsf{CT}_{\mathsf{sym},t'} = \bar{\mathcal{E}}\big(\mathsf{1FE.PK}_1, (\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t'+1}, t', \ell, \mathsf{gsalt}, \sigma_{t'}, \mathsf{Trap}); r_{t'}\big)$$

iii. Set $\mathsf{out}.c_1 = \mathsf{CT}_{\mathsf{sym},t'}$.

(d) **Computing Next Encrypted State.** If $\mathsf{out}.c_2 = \perp$, do the following.

i. Derive the randomness at time step $t+1$ as $r_{t+1} = \mathsf{F.Eval}(\mathsf{K}_{t+1}, (t+1 \| \mathsf{rand} \| \mathsf{salt}))$ and compute the $\mathsf{1FE}_2$ ciphertext component encoding the state $\mathsf{q}_{t+1} = \mathsf{inp}.u_2$ for time step $t+1$ as

$$\mathsf{CT}_{\mathsf{st},t+1} = \bar{\mathcal{E}}\big(\mathsf{1FE}_2.\mathsf{PK}_2, (\mathsf{ST}, \mathsf{q}_{t+1}, \mathsf{gsalt}); r_{t+1}\big)$$

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

(e) **Output :** $\mathsf{out} = \big(c_1, c_2\big)$

Figure 2.9: Function to mimic TM computation. It reads the current symbol, state pair and outputs an encryption of the new state and symbol to be written under the appropriate randomness generated using a cPRF.

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

4. The aggregated sequence of ciphertexts under the Next scheme, of length $L_k = \sum_{j=1}^{k} \ell_j$ computed above is expressed as:
$\big((\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}), \mathsf{CT}_{\mathsf{sym},2}, \ldots, \mathsf{CT}_{\mathsf{sym},\ell_1}, \mathsf{CT}_{\mathsf{sym},\ell_1+1}, \ldots, \mathsf{CT}_{\mathsf{sym},L_k}\big)$.
5. Let $t = 1$. While the Turing machine does not halt, do:

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

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

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

   (b) If $\big((\mathsf{Trap.ST\ TS} = t) \wedge (\mathsf{Trap.Target\ TS} = t+1) \wedge (t > 1)\big)$, compute the $1\mathsf{FE}_2$ state ciphertext $\mathsf{CT}_{\mathsf{st},t+1} = \mathsf{SKE.Dec}(\mathsf{Trap.SKE.K}, \mathcal{C}_2)$ and set $\mathsf{out}.c_2 = \mathsf{CT}_{\mathsf{st},t+1}$.

2. If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_1 = 1)\big)$, do the following:
   (a) If $\big((\mathsf{Trap.Sym\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_1$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_1$ to be encrypted and given as output for time step t'.

   (b) If $\big((\mathsf{Trap.ST\ TS}_1 = t) \wedge (\mathsf{Trap.Target\ TS}_1 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_1$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_1$ to be encrypted and given as output for time step $t+1$.

3. If $\big((\mathsf{Trap.key\text{-}id} = \mathsf{salt}) \wedge (\mathsf{Trap.global\text{-}salt} = \mathsf{gsalt}) \wedge (\mathsf{Trap.mode\text{-}trap}_2 = 1)\big)$, do the following:
   (a) If $\big((\mathsf{Trap.Sym\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t') \wedge (t > \ell)\big)$, set $\mathsf{inp}.u_1 = \mathsf{Trap.Sym\ val}_2$ with the symbol $\sigma_{t'} = \mathsf{Trap.Sym\ val}_2$ to be encrypted and given as output for time step t'.

   (b) If $\big((\mathsf{Trap.ST\ TS}_2 = t) \wedge (\mathsf{Trap.Target\ TS}_2 = t+1) \wedge (t > 1)\big)$, set $\mathsf{inp}.u_2 = \mathsf{Trap.ST\ val}_2$ with the state $\mathsf{q}_{t+1} = \mathsf{Trap.ST\ val}_2$ to be encrypted and given as output for time step $t+1$.

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

Figure 2.10: Subroutine handling the trapdoor modes in Next. This is “active" only in the proof.

(a) Invoke $1\mathsf{FE.Dec}\big(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},t}, \mathsf{CT}_{\mathsf{st},t})\big)$ to obtain:

- ACC or REJ. In this case, output “Accept” or “Reject” respectively, and exit the loop.

- $(\mathsf{CT}_{\mathsf{sym},t'}, \mathsf{CT}_{\mathsf{st},t+1})$.

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

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

2.8.2 Correctness and Efficiency of Multi-Input TMFE

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

Correctness of the second part corresponds to the correct execution of the Turing machine on the aggregate sequence of ciphertexts, and this is exactly the same as in Section 2.7. As before, we maintain the invariant that at any time step t, the input to the $\mathsf{1FE.Dec}$ algorithm is a complete $\mathsf{1FE}$ ciphertext decomposed into two components corresponding to symbol and state (along with additional auxiliary inputs), both computed with the same randomness $\mathsf{F.Eval}(\mathsf{K}_0, (\mathsf{t}\|\mathsf{rand}\|\mathsf{gsalt}))$.

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

1. Invoking $\mathsf{kFE.Dec}$ on the ciphertext $\mathsf{CT}_{1,\mathsf{SYM},1}$ encoding the first symbol of $\mathbf{w}_1$ along with the special ciphertexts $\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}$ encoding $|\mathbf{w}_{\mathsf{ind}}|$ for $\mathsf{ind} \neq 1$ gives $(\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1})$. By correctness of kFE decryption, we have:
$$\mathsf{kFE.Dec}\left(\mathsf{SK}_{\mathsf{Agg}}, \left(\mathsf{CT}_{1,\mathsf{SYM},1}, \{\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}\}_{\mathsf{ind}\in[k]\setminus\{1\}}\right)\right) = \left(\mathsf{CT}_{\mathsf{sym},1}, \mathsf{CT}_{\mathsf{st},1}\right).$$
2. Invoking $\mathsf{kFE.Dec}$ on the ciphertext $\mathsf{CT}_{1,\mathsf{SYM},j}$ encoding the jth symbol of $\mathbf{w}_1$ along with the special ciphertexts $\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}$ encoding $|\mathbf{w}_{\mathsf{ind}}|$ for $\mathsf{ind} \neq 1$ gives

$(\mathsf{CT}_{\mathsf{sym},j}, \perp)$. By correctness of kFE decryption, we have:

$$\mathsf{kFE.Dec}\left(\mathsf{SK}_{\mathsf{Agg}}, \left(\mathsf{CT}_{1,\mathsf{SYM},j}, \{\mathsf{CT}_{\mathsf{ind},\mathsf{SP}}\}_{\mathsf{ind}\in[k]\setminus\{1\}}\right)\right) = (\mathsf{CT}_{\mathsf{sym},j}, \perp).$$

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

$$\mathsf{kFE.Dec}\left(\mathsf{SK}_{\mathsf{Agg}}, \left(\mathsf{CT}_{\mathsf{ind},\mathsf{SYM},j}, \{\mathsf{CT}_{\mathsf{ind}',\mathsf{SP}}\}_{\mathsf{ind}'\in[k]\setminus\{\mathsf{ind}\}}\right)\right) = \left(\mathsf{CT}_{\mathsf{sym},\widetilde{L_i}+j}, \perp\right),$$

where $\widetilde{L_i} = \sum_{m=1}^{\mathsf{ind}-1} \ell_m$.

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

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

1. Let $t \in [\tau - 2] \setminus [\ell]$. If the current work tape cell was accessed[15], at some time step $\tilde{t} < t$, then $\mathsf{CT}_{\mathsf{sym},t}$ encoding $(\mathsf{SYM}, \mathsf{key\text{-}id}, \mathsf{K}_{t+1}, t, \ell, \mathsf{gsalt}, \sigma_t, \mathsf{Trap})$ was constructed at time step $\tilde{t}$. Note that σ_t may be the blank symbol β. When $t \in [\ell]$, $\mathsf{CT}_{\mathsf{sym},t}$ is constructed at time step t via the Agg circuit.
2. The ciphertext component $\mathsf{CT}_{\mathsf{st},t}$ encoding $(\mathsf{ST}, \mathsf{q}_t, \mathsf{gsalt})$ at time step t was constructed at time step $t-1$ for $t > 1$ and at time step 1, when $t = 1$.
3. The randomness $r_t = \mathsf{F.Eval}(\mathsf{K}_{\tilde{\mathsf{t}}+1}, (\mathsf{t}\|\mathsf{rand}\|\mathsf{gsalt})) = \mathsf{F.Eval}(\mathsf{K}_{\mathsf{t}}, (\mathsf{t}\|\mathsf{rand}\|\mathsf{gsalt}))$ binds $\mathsf{CT}_{\mathsf{sym},t}$ and $\mathsf{CT}_{\mathsf{st},t}$ and both the encoded messages also share the same global salt.

Thus, at any given time step $t \in [\tau - 2]$, we have a complete ciphertext of 1FE which may be fed again with $\mathsf{SK}_{\mathsf{Next}}$ to 1FE.Dec in order to proceed with the computation. Thus, the execution of 1FE.Dec at the $(\tau-2)^{\text{th}}$ time step provides the complete pair $(\mathsf{CT}_{\mathsf{sym},\tau-1}, \mathsf{CT}_{\mathsf{st},\tau-1})$. By the correctness of 1FE scheme again, at time step $t = \tau - 1$,

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

invoking $\mathsf{1FE.Dec}(\mathsf{SK}_{\mathsf{Next}}, (\mathsf{CT}_{\mathsf{sym},\tau-1}, \mathsf{CT}_{\mathsf{st},\tau-1}))$ outputs either “Accept" or “Reject" by simulating the execution of M for the final time step τ inside the function Next, thus correctly outputting $M(\mathbf{w})$.

Efficiency. The kTMFE construction described above inherits its efficiency from the underlying kFE and $\mathsf{1FE}$. The ciphertext is compact and is of size $\mathrm{poly}(\lambda, k, |\mathbf{w}|)$, where $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_k)$. The running time of the decryption procedure is input specific since it mimics the computation of M on $\mathbf{w}$ using secret key encoding M and ciphertext encoding all the intermediate states of the computation. Additionally, kFE and $\mathsf{1FE}$ being compact CktFE schemes, the parameters are short $\mathrm{poly}(k, \lambda)$ and $\mathrm{poly}(\lambda)$ respectively. The function keys are also short, since they are CktFE function keys for circuits Agg and Next which are of size $\mathrm{poly}(k, \lambda)$ and $\mathrm{poly}(|M|, \lambda)$ respectively.

2.8.3 Proof of Security for Multi-Input TMFE

Security of the above construction follows the same blueprint as the proof in Section 2.7 except that instead of single input functionality ReRand, we now use a k-input functionality Agg to aggregate and rerandomize the inputs. We emphasize that the outputs produced by the Agg functionality are *exactly the same* as the outputs produced by ReRand functionality in Section 2.7: namely a sequence of $\mathsf{1FE}$ ciphertexts encoding the symbol and global position, computed using randomness derived from a cPRF. Hence, the chief new ingredient in the security proof is the security of Agg functionality, which is derived from the security of the kFE scheme.

Formally, we argue that:

Theorem 2.8.1. *Assume that the k input FE for circuits kFE satisfies standard indistinguishability (Definition 2.6.6), and the single input FE for circuits $\mathsf{1FE}$ satisfies distributional indistinguishability (Definition 2.6.4). Assume that the cPRF is secure according to definition 2.6.13. Then, the above construction of k input kTMFE satisfies standard indistinguishability (Definition 2.6.9).*

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

In the single input case, we replaced the execution chain of a machine over an input string from $b = 0$ to $b = 1$, step by step, and enumerated over all keys. Now, we again replace an execution chain step by step as in the single input case, but additionally enumerate over all Q^k combinations for each key, as well as over all keys as before. The number of hybrids grows multiplicatively by Q^k. Since the proof structure follows mostly the single input case, we provide only the conceptual description of the main ideas and the hybrids' sequence in the proof in Appendix A.2.

2.9 Indistinguishability Obfuscation for Turing Machines

In this section we construct indistinguishability obfuscation for Turing machines with bounded length input, i.e. the input length $n = n(\lambda)$ is any fixed polynomial in the security parameter.

Our construction is a straightforward adaptation of the miFE to iO compiler for circuits [Goldwasser *et al.* (2014)] to Turing machines. To support inputs of length n, we need an $(n+1)$-ary miFE for Turing machines denoted as $(\mathsf{n}{+}1)$-TMFE; we instantiate this with our construction from Section 2.8.

In more detail, the obfuscation of M comprises the secret key $\mathsf{SK_U}$ for the Universal Turing machine and $(2n+1)$ ciphertexts under the $(n+1)$-TMFE scheme, where the first $2n$ ciphertexts $\{\mathsf{CT}_i^b\}_{i\in[n],b\in\{0,1\}}$ encode bits 0 and 1 respectively for each of n positions while the last ciphertext CT_M encodes machine $\langle M\rangle$. To evaluate $\mathsf{iO}(M)$ on an input $\mathbf{x} = (x_1, \dots, x_n) \in \Sigma_\lambda^n$, the evaluator runs $(\mathsf{n}{+}1)$-$\mathsf{TMFE.Dec}\big(\mathsf{SK_U}, (\{\mathsf{CT}_i^{x_i}\}_{i\in[n]}, \mathsf{CT}_M)\big)$ to get $M(\mathbf{x})$. To argue security we only need the $(\mathsf{n}{+}1)$-TMFE scheme to be selectively secure against two ciphertext queries per slot and a single key query, as in the case of circuits.

2.9.1 Construction

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

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

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

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

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

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

Correctness and Efficiency. Correctness is directly followed by the correctness of $(\mathsf{n{+}1})\text{-}\mathsf{TMFE}$ scheme. Since the $(\mathsf{n{+}1})\text{-}\mathsf{TMFE}$ we use is compact, the obfuscation size obtained by the above scheme is $\mathsf{poly}(\lambda, |\mathsf{U}|, |M|, n)$.

2.9.2 Proof of Security

We show that the construction is secure. Formally:

Theorem 2.9.1. *Assume that* $(\mathsf{n}{+}1)$-TMFE *is a* 1*-key,* 2*-ciphertext selectively secure* $(n+1)$*-ary multi-input functional encryption scheme for Turing machines which satisfies standard indistinguishability (Section 2.6.3). Then the construction in Section 2.9.1 is a secure indistinguishability obfuscator for the Turing machines (Section 2.6.3) with bounded input length* n*.*

Proof. Consider two Turing machines $M_0, M_1 \in \mathcal{M}_\lambda$ such that $\forall \mathbf{x} \in \Sigma_\lambda^n, M_0(\mathbf{x}) = M_1(\mathbf{x})$. We now show that if there exists a PPT adversary $\mathcal{A}$ that distinguishes between $\widetilde{M_0} = \mathsf{iO}(M_0)$ and $\widetilde{M_1} = \mathsf{iO}(M_1)$ with non-negligible advantage, then there exists another PPT adversary $\mathcal{B}$ which breaks the $(\mathsf{n}{+}1)$-TMFE scheme with the same advantage. We construct $\mathcal{B}$ as follows.

$\mathcal{B}$ runs $\mathcal{A}$ to get two functionally equivalent machines $M_0, M_1 \in \mathcal{M}_\lambda$. It does the following:

1. $\mathcal{B}$ prepares a pair of sequences $(\mathbf{x}^0, \mathbf{x}^1)$, each containing two challenge message vectors for the $(\mathsf{n}{+}1)$-TMFE challenger $\mathcal{C}$ such that for all $b \in \{0,1\}, \mathbf{x}^b = \big\{(x^b_{1,1}, \ldots, x^b_{n+1,1}), (x^b_{1,2}, \ldots, x^b_{n+1,2})\big\}$.
   - For all $i \in [n]$, $\mathcal{B}$ sets $x^0_{i,1} = x^1_{i,1} = 0$ and $x^0_{i,2} = x^1_{i,2} = 1$
   - For $i = n+1$, $\mathcal{B}$ sets $x^b_{n+1,1} = x^b_{n+1,2} = \langle M_b \rangle$, where $\mathsf{Encode}_\lambda(\mathsf{M_b}) = \langle \mathsf{M_b} \rangle$.

   $\mathcal{B}$ sends the pair $(\mathbf{x}^0, \mathbf{x}^1)$ to $\mathcal{C}$ and receives $(\mathsf{CT}_{1,j}, \ldots, \mathsf{CT}_{n+1,j})_{j\in[2]}$.
2. $\mathcal{B}$ requests $\mathcal{C}$ for a secret key corresponding to machine U_λ and receives SK_U.
3. $\mathcal{B}$ sends $\widetilde{M} = (\mathsf{SK}_\mathsf{U}, (\{\mathsf{CT}_{1,j}, \ldots, \mathsf{CT}_{n,j}\}_{j\in[2]}, \mathsf{CT}_{n+1,1}))$ as the challenge obfuscation to $\mathcal{A}$ and outputs a bit b' returned by $\mathcal{A}$.

This completes the description of the reduction $\mathcal{B}$. We first observe that for any $\mathbf{x} = (x_1, \ldots, x_n) \in \Sigma_\lambda^n$, since M_0 and M_1 are functionally equivalent Turing machines, we have that:

$$\mathsf{U}_\lambda(\mathbf{x}, \langle M_0 \rangle) = M_0(\mathbf{x}) = M_1(\mathbf{x}) = \mathsf{U}_\lambda(\mathbf{x}, \langle M_1 \rangle)$$

Further, $\mathcal{A}$ being a valid iO adversary, we have $\mathsf{runtime}(M_0, \mathbf{x}) = \mathsf{runtime}(M_1, \mathbf{x})$. Thus $\mathcal{B}$ is a valid $(\mathsf{n}{+}1)$-TMFE adversary. Hence, if the $(\mathsf{n}{+}1)$-TMFE challenger had chosen challenge bit 0, then the obfuscation $\widetilde{M}$ is of M_0, else of M_1. Thus the advantage of $\mathcal{A}$ in distinguishing the two cases translates exactly to the advantage of $\mathcal{B}$ against the $(\mathsf{n}+1)$-TMFE scheme. $\square$

CHAPTER 3

Attribute Based Encryption and its Generalizations for Nondeterministic Finite Automata from Lattices

3.1 Introduction

In this chapter we provide new constructions of attribute based encryption (ABE) and its generalizations in the nondeterministic finite automata model of computation. Before stating our results in detail, we first describe ABE informally and discuss below the state of the art prior to our work.

Attribute based encryption (ABE) [Sahai and Waters (2005)] is an encryption paradigm that enables fine grained access control on encrypted data. In attribute based encryption, a ciphertext of a message m is labelled with a public attribute $\mathbf{x}$ and secret keys are labelled with a Boolean function f. Decryption succeeds to yield the hidden message m if and only if the attribute satisfies the function, namely $f(\mathbf{x}) = 1$. Starting with the seminal work of [Sahai and Waters (2005)], ABE schemes have received a lot of attention in recent years [Goyal *et al.* (2006); Boneh and Waters (2007); Bethencourt *et al.* (2007); Katz *et al.* (2008); Lewko *et al.* (2010); Agrawal *et al.* (2011); Waters (2012); Gorbunov *et al.* (2013); Boneh *et al.* (2014); Gorbunov *et al.* (2015); Gorbunov and Vinayagamurthy (2015); Brakerski and Vaikuntanathan (2016); Ananth *et al.* (2019)], yielding constructions for various classes of functions under diverse assumptions.

ABE for uniform models of computation has also been studied, but so far, we have very few constructions from standard assumptions. [Waters (2012)] provided a construction of ABE for Deterministic Finite Automata (DFA) from parametrized or "q-type" assumptions over bilinear maps. Generalizing this construction to Nondeterministic Finite Automata (NFA) was left as an explicit open problem[1] in [Waters (2012)], and

[1]Note that an NFA can be converted to an equivalent DFA but this transformation incurs exponential blowup in machine size.

has remained open to date. Constructions from other assumptions such as more standard pairing based assumptions, or lattice based assumptions has also proved elusive. [Boyen and Li (2015)] provided a construction of ABE for DFA from the Learning With Errors (LWE) assumption but this was restricted to DFAs with *bounded* length inputs, rendering moot the primary advantage of a DFA over circuits. Agrawal and Singh [Agrawal and Singh (2017)] constructed a primitive closely related to ABE for DFA, namely *reusable garbled DFA* from LWE, but their construction is limited to a security game where the adversary may only request a single function key.

From strong assumptions such as the existence of multilinear maps [Garg *et al.* (2013*a*)], witness encryption [Goldwasser *et al.* (2013*b*)] or indistinguishability obfuscation [Barak *et al.* (2001); Garg *et al.* (2013*a*)], attribute based encryption (indeed, even its more powerful generalization – *functional encryption*) has been constructed even for Turing machines [Ananth and Sahai (2017); Agrawal and Maitra (2018); Kitagawa *et al.* (2019)], but these are not considered standard assumptions; indeed many candidate constructions have been broken [Cheon *et al.* (2015); Coron *et al.* (2015); Hu and Jia, (2015); Cheon *et al.* (2016*b*,*a*); Miles *et al.* (2016); Coron *et al.* (2017); Apon *et al.* (2017)]. Very recently, [Ananth *et al.* (2019)] constructed ABE for RAM programs from LWE achieving decryption complexity that is sublinear in the database length. However, the key sizes in their construction are massive and grow with the size of the entire database as well as with worst case running time of the program on any input. In particular, restricting the construction to any model of computation that reads the entire input string (e.g. DFA, TM) yields a bounded input solution, since the key size depends on the input length. Similarly, [Brakerski and Vaikuntanathan (2016); Goyal *et al.* (2016)] construct attribute based encryption for "bundling functionalities" where the size of the public parameters does not depend on the size of the input chosen by the encryptor, say ℓ. However, the key generator must generate a key for a circuit with a fixed input length, say ℓ', and decryption only succeeds if $\ell = \ell'$. Thus, bundling functionalities do not capture the essential challenge of supporting dynamic data sizes; this was noted explicitly in [Goyal *et al.* (2016)].

3.2 Our Contributions

In this work, we construct the first symmetric key attribute based encryption scheme for nondeterministic finite automata (NFA) from the learning with errors (LWE) assumption in lattices. Our scheme supports unbounded length inputs as well as unbounded length machines. In more detail, secret keys in our construction are associated with an NFA M of *unbounded* length, ciphertexts are associated with a tuple $(\mathbf{x}, m)$ where $\mathbf{x}$ is a public attribute of *unbounded* length and m is a secret message bit, and decryption recovers m if and only if $M(\mathbf{x}) = 1$. Moreover our construction achieves succinct parameters, namely, the length of the function key and ciphertext grow only with the machine size and input length respectively (and do not depend on the input length and machine size respectively).

Further, we leverage our ABE to achieve (restricted notions of) attribute hiding analogous to the circuit setting, obtaining the first *predicate encryption* and bounded key *functional encryption* schemes for NFA. We achieve machine hiding in the single key[2] setting to obtain the first *reusable garbled NFA* from standard assumptions. This improves upon the result of [Agrawal and Singh (2017)], which can only support a *single* key request (as against bounded), and only DFAs (as against NFAs).

The above results raise the question of whether full fledged functional encryption (please see Appendix B.1.3 for the formal definition), which achieves full attribute hiding for NFAs is possible under standard assumptions. However, we show that secret key functional encryption even for DFA with security against unbounded key requests implies indistinguishability obfuscation (iO) for circuits. Since constructing iO for circuits from standard assumptions is a central challenge in cryptography, this suggests that there is a barrier in further generalizing our result to achieve full attribute hiding.

We summarize our results in Table 3.1.

3.3 Our Techniques

In this section, we provide an overview of our techniques. Before we proceed, we discuss the technical barriers that arise in following the approaches taken by prior work. Since the construction by [Waters (2012)] is the only one that supports unbounded attribute

[2]This may be generalized to bounded key, for any a-priori fixed (polynomial) bound.

Construction	**Model**	**Input Length**	**Number of Keys**	**Attribute and Function Hiding**	**Assumption**
[Waters (2012)]	DFA	unbounded	unbounded	(no, no)	q-type assumption on bilinear maps
[Boyen and Li (2015)]	DFA	bounded	unbounded	(no, no)	LWE
[Agrawal and Singh (2017)]	DFA	unbounded	single	(yes, yes)	LWE
[Ananth *et al.* (2019)]	RAM	bounded	unbounded	(no, no)	LWE
Section 3.7	NFA	unbounded	unbounded	(no, no)	LWE
Appendix B.2	NFA	unbounded	unbounded	(yes*, no)	LWE
Appendix B.4	NFA	unbounded	bounded	(yes, yes)	LWE

Table 3.1: Prior work and our results. Above, we say that input length supported by a construction is bounded if the parameters and key lengths depend on the input size. For attribute hiding, yes* indicates hiding in the restricted security games of predicate or bounded key functional encryption.

lengths and unbounded key requests by the adversary, [3] it is the most promising candidate for generalization to NFA. However, the challenges in generalizing this construction to support NFAs were explicitly discussed in the same work, and this has seen no progress in the last seven years to the best of our knowledge, despite the significant research attention ABE schemes have received. Moreover, even the solution for DFAs is not fully satisfactory since it relies on a non-standard parametrized or "q-type" assumption.

Boyen and Li [Boyen and Li (2015)] attempt to construct ABE for DFAs from the LWE assumption, but their construction crucially requires the key generator to know the length of the attribute chosen by the encryptor, since it must provide a fresh "trapdoor" for each row of the DFA transition table and each input position. Indeed, reusing the same trapdoor for multiple positions in the input leads to trivial "mix and match" attacks against their scheme. Thus, it is not even clear how to obtain ABE for DFA with support for unbounded lengths by following this route. The work of Agrawal and Singh [Agrawal

[3]The construction is later extended to be adaptively secure rather than selectively secure (e.g., [Attrapadung (2014)]), but the basic structure of the construction is unchanged.

and Singh (2017)] gives a construction of functional encryption for DFA from LWE that does handle unbounded length inputs, but only in the limited single key setting. Their construction crucially relies on reusable garbled circuits [Goldwasser *et al.* (2013*a*)] which is a single key primitive, and natural attempts to generalize their construction to support even two keys fails[4]. Similarly, the very recent construction of [Ananth *et al.* (2019)] is also inherently bounded length, for reasons similar as those discussed above for [Boyen and Li (2015)].

Thus, the handful of existing results in this domain all appear to pose fundamental barriers to generalization. To overcome this hurdle, we design completely new techniques to handle the challenge of unbounded length; these may be applicable elsewhere. We focus on the symmetric key setting, and proceed in two steps: i) we provide a secret key ABE scheme for NFA that supports unbounded length inputs but only supports bounded size NFA machines, and ii) we "bootstrap" the construction of step (i) to handle unbounded length machines. We additionally achieve various notions of attribute hiding as discussed above, but will focus on the ABE construction for the remainder of this overview. We proceed to describe each of these steps in detail.

Constructing NfaABE for Bounded Size NFA. Our first goal is to construct a secret key ABE scheme for NFA that supports unbounded length inputs but only supports bounded size NFA machines from the LWE assumption. Since ABE for circuits has received much success from the LWE assumption [Gorbunov *et al.* (2013); Boneh *et al.* (2014)], our first idea is to see if we can run many circuit ABE schemes "in parallel", one for each input length. We refer to our resulting ABE scheme for NFAs as NfaABE and the ABE for circuits scheme simply as ABE, in order to differentiate them.

Naïve Approach: We start with the following naïve construction that uses a (public key) ABE for circuits as an ingredient. The master secret key of the NfaABE scheme is a PRF key K. This PRF key defines a set of key pairs $\{(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j)\}_{j\in[2^\lambda]}$ of the ABE scheme, where each $(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j)$ is sampled using randomness derived from the PRF key K and supports circuits with inputs of length j. When one encrypts a message for a ciphertext attribute $\mathbf{x}$, one chooses the master public key $\mathsf{ABE.mpk}_{|\mathbf{x}|}$ and encrypts the message using the key, where $|\mathbf{x}|$ is the length of $\mathbf{x}$. We can

[4]For the knowledgeable reader, bounded key variants of reusable garbled circuits exist, for instance by applying the compiler of [Gorbunov *et al.* (2012)], but using this in the aforementioned construction does not work due to the structure of their construction.

encrypt for $\mathbf{x}$ with length at most 2^λ and therefore can deal with essentially unbounded length strings as ciphertext attributes. In order to generate a secret key for a machine M, one has to convert it into a circuit since our underlying ingredient is an ABE for circuits. The difference between an NFA machine M and a circuit is that while the former takes a string with arbitrary length as an input, the input length for the latter is fixed. To fill the gap, we prepare a circuit version of NFA M for all possible input lengths. Namely, we convert the machine M into an equivalent circuit $\widehat{M_j}$ with input length j for all $j \in [2^\lambda]$. Then, we generate ABE secret key associated with $\widehat{M_j}$ by running the key generation algorithm of the ABE for all j to obtain the NfaABE secret key $\{\mathsf{ABE.sk}_j\}_{j\in[2^\lambda]}$. When decrypting a ciphertext associated with $\mathbf{x}$, the decryptor chooses $\mathsf{ABE.sk}_{|\mathbf{x}|}$ and runs the decryption algorithm of the underlying ABE to retrieve the message.

Reducing the Number of Keys: Obviously, there are multiple problems with this approach. The first problem is that there are 2^λ instances of ABE and thus the secret key of NfaABE is exponentially large. To handle this, we thin out most of the instances and change the secret key to be $\{\mathsf{ABE.sk}_{2^j}\}_{j\in[0,\lambda]}$. In order to make sure that the decryption is still possible even with this change, we modify the encryption algorithm. To encrypt a message for an attribute $\mathbf{x}$, one chooses $i \in [0,\lambda]$ such that $2^{i-1} < |\mathbf{x}| \leq 2^i$ and uses the i-th instance to encrypt the message, where if the length of $\mathbf{x}$ is not exactly 2^i, it is padded with blank symbols to adjust the length. This change reduces the number of instances down to be polynomial.

Reducing the Size of Keys: However, a bigger problem is that even though we reduced the *number* of secret keys, we did not reduce their size, which is still not polynomial. In particular, there is no guarantee on the size of $\mathsf{ABE.sk}_{2^\lambda}$ since the associated circuit $\widehat{M}_{2^\lambda}$ is of exponential size. Here, we leverage a crucial efficiency property that is enjoyed by the ABE for circuits constructed by [Boneh *et al.* (2014)], namely, that the secret keys in this scheme are very short. The size of secret keys in their scheme is dependent only on the depth of the circuits being supported and *independent of the input length and size*. Thus, if we can ensure that the depth of $\widehat{M}_{2^\lambda}$ is polynomially bounded (even though the input is exponentially long), we are back in business.

However, converting the NFA to a circuit requires care. We note that implementing the trivial approach of converting an NFA to a circuit by keeping track of all possible states while reading input symbols results in circuit whose depth is linear in input length, which is exponential. To avoid this, we make use of a divide and conquer approach to evaluate the NFA, which makes the circuit depth poly-logarithmic in the input length. As a result, the size of the secret keys can be bounded by a polynomial as desired.

Efficiency of Key Generation: The final and the most difficult problem to be addressed is that even though we managed to make the size of $\{\mathsf{ABE.sk}_{2^j}\}_{j\in[0,\lambda]}$ polynomially bounded, computational time for generating it is still exponentially large, since so is the size of the associated circuits $\{\widehat{M_{2^j}}\}_{j\in[0,\lambda]}$. To resolve the problem, we note that the only algorithm which has the "space" to handle the unbounded input length is the encryption algorithm. Hence, we carefully divide the computation of generating $\{\mathsf{ABE.sk}_{2^j}\}_{j\in[0,\lambda]}$ into pieces so that the key generator only needs to do work proportional to the size of the machine, the encryptor does work proportional to the size of the input and the decryptor computes the requisite key on the fly.

To implement this idea, we use succinct single-key functional encryption (FE), which can be realized from the LWE assumption [Goldwasser *et al.* (2013*a*); Agrawal (2017)]. To support unbounded input length, we generate $\lambda + 1$ instances of the FE scheme to obtain $\{\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j\}_{j\in[0,\lambda]}$. The secret key of NfaABE is $\{\mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]}$, where $\mathsf{FE.ct_j} = \mathsf{FE.Enc}(\mathsf{FE.mpk_j}, (\mathsf{M}, \mathsf{K}))$ is an encryption of a description of the associated NFA M and the PRF key K under the j-th instance of the FE scheme. To provide the matching secret key, the encryptor appends $\mathsf{FE.sk}_i = \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, C_i)$ to the ciphertext. Here, $\mathbf{x}$ is the attribute vector of unbounded length, i is an integer such that $2^{i-1} < |\mathbf{x}| \le 2^i$ and C_i is a circuit that takes as inputs the machine M and PRF key K and outputs an ABE secret key $\mathsf{ABE.sk}_{2^i}$ associated with M. While this step helps the decryptor to compute the ABE secret key on the fly, it also limits the current construction to the secret key setting crucially.

We are almost done – the decryptor chooses $\mathsf{FE.ct_i}$ with appropriate i from the received set $\{\mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]}$ and decrypts it using $\mathsf{FE.sk}_i$ that is appended to the ciphertext to obtain an ABE secret key $\mathsf{ABE.sk}_{2^i}$. Then, it decrypts the ABE ciphertext also provided in the ciphertext to retrieve the message. Note that our construction is carefully designed so that we only require a *single* key of the succinct FE scheme.

Arguing the efficiency of the scheme requires care. In order to make the key generation algorithm run in polynomial time, we rely on the succinctness of the underlying FE. Recall that the succinctness property says that the running time of the encryption algorithm is independent of the size of the circuits being supported and only dependent on the depth and input and output length. In our construction, the computation of $\{\mathsf{FE.ct_j} = \mathsf{FE.Enc}(\mathsf{FE.mpk_j}, (\mathsf{M}, \mathsf{K}))\}_{\mathsf{j} \in [0,\lambda]}$ can be performed in polynomial time, since the input length $|M| + |\mathsf{K}|$ is bounded by a fixed polynomial[5] and so is the output length $|\mathsf{ABE.sk}_{2^j}|$. Note that we crucially use the succinctness of the FE here, since the size of the circuit C_{2^j}, which is supported by the j-th instance of FE, is polynomial in 2^j and thus exponential for $j = \lambda$.

Security: Our construction of NfaABE satisfies standard (selective) indistinguishability based security. The high level idea of the proof is outlined next. Intuitively, security follows from the security of the single key FE scheme and the underlying circuit ABE scheme. In the first step, we show that even though an adversary can obtain multiple FE ciphertexts and secret keys, it cannot obtain anything beyond their decryption results $\{\mathsf{FE.Dec}(\mathsf{FE.sk}_i, \mathsf{FE.ct_i}) = \mathsf{ABE.sk_i}\}$ by the security of the FE. Then, we leverage the security of the ABE to conclude that the message is indeed hidden. We note that in order to invoke the FE security, we need to ensure that only one secret key is revealed to the adversary for each instance of FE. This property is guaranteed, since the circuit for which a secret key of the j-th instance of FE is generated is fixed (i.e., C_{2^j}). Please see Section 3.6 for details.

Removing the Size Constraint on NFAs. So far, we have constructed NfaABE for NFA that can deal with unbounded input length and bounded size NFAs. Let us call such a scheme (u, b)-NfaABE, where “u" and “b" stand for “unbounded" and “bounded". We define (b, u)-NfaABE and (u, u)-NfaABE analogously, where the first parameter refers to input length and the second to machine size. Our goal is to obtain (u, u)-NfaABE. At a high level, we compile (u, u)-NfaABE using two pieces, namely (u, b)-NfaABE which we have already constructed, and (b, u)-NfaABE, which we will instantiate next.

To construct (b, u)-NfaABE, our basic idea is to simply convert an NFA into an equivalent circuit and then use existing ABE for circuits schemes [Gorbunov *et al.* (2013); Boneh *et al.* (2014)]. This approach almost works, but we need to exercise care to ensure that the depth of these circuits can be bounded since we hope to support NFAs of unbounded size. To fill this gap, we show that an NFA can be converted into an equivalent circuit whose depth is poly-logarithmic in the size of the NFA by again using the divide and conquer approach we discussed previously. This enables us to

[5]Recall that we are only dealing with bounded size NFAs.

bound the depth of the circuits by a fixed polynomial, even if the size of corresponding NFA is unbounded and allows us to use existing ABE schemes for circuits to construct (b,u)-NfaABE.

We are ready to construct (u,u)-NfaABE by combining (u,b)-NfaABE and (b,u)-NfaABE. The master secret key of the (u,u)-NfaABE is a PRF key K. This PRF key defines a set of keys $\{(\mathsf{u},\mathsf{b})\text{-}\mathsf{NfaABE.msk}_j\}_{j\in[2^\lambda]}$ of the (u,b)-NfaABE scheme, where each (u,b)-$\mathsf{NfaABE.msk}_j$ supports NFAs with size j. Similarly, the PRF key also defines keys $\{(\mathsf{b},\mathsf{u})\text{-}\mathsf{NfaABE.msk}_j\}_{j\in[2^\lambda]}$ of the (b,u)-NfaABE scheme, where each (b,u)-$\mathsf{NfaABE.msk}_j$ supports input strings with length j. To encrypt a message with respect to a ciphertext attribute $\mathbf{x}$, it encrypts the message for $\mathbf{x}$ using (u,b)-$\mathsf{NfaABE.msk}_j$ to obtain (u,b)-$\mathsf{NfaABE.ct_j}$ for all $j\in[\mathbf{x}]$. Furthermore, it also encrypts the message for $\mathbf{x}$ using (b,u)-$\mathsf{NfaABE.msk}_{|\mathbf{x}|}$ to obtain (b,u)-$\mathsf{NfaABE.ct}_{|\mathbf{x}|}$. The final ciphertext is

$$\left(\ \{(\mathsf{u},\mathsf{b})\text{-}\mathsf{NfaABE.ct_j}\}_{\mathsf{j}\in[|\mathbf{x}|]},\ (\mathsf{b},\mathsf{u})\text{-}\mathsf{NfaABE.ct}_{|\mathbf{x}|}\ \right).$$

To generate a secret key for a machine M, we essentially swap the roles of (u,b)-NfaABE and (b,u)-NfaABE. Namely, we generate a secret key (b,u)-$\mathsf{NfaABE.sk}_j$ for M using (b,u)-$\mathsf{NfaABE.msk}_j$ for all $j\in[|M|]$, where $|M|$ is the size of the machine M. We also generate (u,b)-$\mathsf{NfaABE.sk}_{|M|}$ for M using (u,b)-$\mathsf{NfaABE.msk}_{|M|}$. The final secret key is

$$\left(\ (\mathsf{u},\mathsf{b})\text{-}\mathsf{NfaABE.sk}_{|M|},\ \{(\mathsf{b},\mathsf{u})\text{-}\mathsf{NfaABE.sk}_j\}_{j\in[|M|]}\ \right).$$

To decrypt a ciphertext for attribute $\mathbf{x}$ using a secret key for an NFA machine M, we first compare $|\mathbf{x}|$ and $|M|$. If $|\mathbf{x}| > |M|$, it decrypts (u,b)-$\mathsf{NfaABE.ct_{|M|}}$ using (u,b)-$\mathsf{NfaABE.sk}_{|M|}$. Otherwise, it decrypts (b,u)-$\mathsf{NfaABE.ct}_{|\mathbf{x}|}$ using (u,b)-$\mathsf{NfaABE.sk}_{|\mathbf{x}|}$. It is not hard to see that the correctness of the resulting scheme follows from those of the ingredients. Furthermore, the security of the scheme is easily reduced to those of the ingredients, as the construction simply runs them in parallel with different parameters. The proof is by a hybrid argument, where we change the encrypted messages in a instance-wise manner. In Sec. 3.7, we streamline the construction and directly construct (u,u)-NfaABE from (u,b)-NfaABE and ABE for circuits instead of going through (b,u)-NfaABE.

Generalizations and Lower Bounds. We further generalize our ABE construction to obtain predicate encryption and bounded key functional encryption for NFAs along with the first construction of reusable garbled NFA. These constructions are obtained by carefully replacing the underlying ABE for circuits with predicate encryption, bounded key functional encryption for circuits or reusable garbled circuits. This compiler requires some care as we need to argue that the delicate balance of efficiency properties that enable our NfaABE construction are not violated, as well as ensure that the constructions and security proofs translate. In Appendix B.2 and Appendix B.4, we show that we can indeed ensure this, sometimes (see for instance, the construction in Appendix B.4) by employing additional tricks as required. In Section 3.8 we show that secret key functional encryption (SKFE) for DFA with security against unbounded collusion implies indistinguishability obfuscation for circuits. There, we essentially show that we can convert an SKFE for DFA into an SKFE for NC_1 circuit, which implies indistinguishability obfuscation for circuits by previous results [Ananth *et al.* (2015*a*); Kitagawa *et al.* (2018*a*)]. The conversion is by encoding and purely combinatorial – we first convert an NC_1 circuit into an equivalent branching program and then leverage the similarity between the branching program and DFA to obtain the result.

3.4 Organization

We organize the rest of the chapter as follows. In Section 3.5, we provide the definitions and preliminaries we require. In Section 3.6, we provide our ABE for NFA supporting unbounded input but bounded machine length. In Section 3.7, we enhance the construction to support both unbounded input and unbounded machine length. In Appendix B.2 we leverage our ABE to construct the first predicate and bounded key functional encryption schemes for NFA. In Appendix B.4, we provide the first construction of reusable garbled NFA. In Section 3.8 we show that secret key functional encryption for DFA with security against unbounded collusion implies indistinguishability obfuscation for circuits.

3.5 Preliminaries

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

Notation. For a circuit $C : \{0,1\}^{\ell_1+\ell_2} \to \{0,1\}$ and a string $\mathbf{x} \in \{0,1\}^{\ell_1}$, $C[\mathbf{x}] : \{0,1\}^{\ell_2} \to \{0,1\}$ denotes a circuit that takes $\mathbf{y}$ and outputs $C(\mathbf{x},\mathbf{y})$. We construct $C[\mathbf{x}]$ in the following specified way. Namely, $C[\mathbf{x}]$ is the circuit that takes as input $\mathbf{y}$ and sets

$$z_i = \begin{cases} y_1 \wedge \neg y_1 & \text{if } x_i = 0 \\ y_1 \vee \neg y_1 & \text{if } x_i = 1 \end{cases}$$

and then computes $C(\mathbf{z},\mathbf{y})$, where x_i, y_i, and z_i are the i-th bit of $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, respectively. In the above, it is clear that $z_i = x_i$ and we have $C(\mathbf{z},\mathbf{y}) = C(\mathbf{x},\mathbf{y})$. Furthermore, it is also easy to see that $\mathsf{depth}(C[\mathbf{x}]) \le \mathsf{depth}(C) + O(1)$ holds.

3.5.1 Definitions: Non Deterministic Finite Automata

A Non-Deterministic Finite Automaton (NFA) M is represented by the tuple $(Q, \Sigma, T, q_{\mathsf{st}}, F)$ where Q is a finite set of states, Σ is a finite alphabet, $T : \Sigma \times Q \to \mathcal{P}(Q)$ is the transition function (stored as a table), q_{st} is the start state, $F \subseteq Q$ is the set of accepting states. For states $q, q' \in Q$ and a string $\mathbf{x} = (x_1, \ldots, x_k) \in \Sigma^k$, we say that q' is reachable from q by reading $\mathbf{x}$ if there exists a sequence of states $q_1, \ldots, q_{k+1}$ such that $q_1 = q_{\mathsf{st}}$, $q_{i+1} \in T(x_i, q_i)$ for $i \in [k]$ and $q_{k+1} = q'$. We say $M(\mathbf{x}) = 1$ iff there is a state in F that is reachable from q_{st} by reading $\mathbf{x}$.

Remark 3.5.1. As it is known, we can transform an NFA with ϵ-transitions into a one without them by a simple and efficient conversion. The conversion preserves the size of the NFA. For simplicity and without loss of generality, we do not deal with an NFA with ϵ-transitions in this chapter.

3.5.2 Definitions: Secret-key Attribute Based Encryption for NFA

A secret-key attribute-based encryption (SKABE) scheme NfaABE for a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four algorithms. In the following, we fix some alphabet

$\Sigma = \Sigma_\lambda$ of size $2 \le |\Sigma| \le \mathrm{poly}(\lambda)$.

- $\mathsf{NfaABE.Setup}(1^\lambda)$ is a PPT algorithm takes as input the unary representation of the security parameter and outputs the master secret key $\mathsf{NfaABE.msk}$.
- $\mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \mathbf{x}, m)$ is a PPT algorithm that takes as input the master secret key $\mathsf{NfaABE.msk}$, a string $\mathbf{x} \in \Sigma^*$ of arbitrary length and a message $m \in \mathcal{M}$. It outputs a ciphertext $\mathsf{NfaABE.ct}$.
- $\mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk}, M)$ is a PPT algorithm that takes as input the master secret key $\mathsf{NfaABE.msk}$ and a description of an NFA machine M. It outputs a corresponding secret key $\mathsf{NfaABE.sk}_M$.
- $\mathsf{NfaABE.Dec}(\mathsf{NfaABE.sk}_M, M, \mathsf{NfaABE.ct}, \mathbf{x})$ is a deterministic polynomial time algorithm that takes as input the secret key $\mathsf{NfaABE.sk}_M$, its associated NFA M, a ciphertext $\mathsf{NfaABE.ct}$, and its associated string $\mathbf{x}$ and outputs either a message m' or $\perp$.

Remark 3.5.2. In our construction in Sec. 3.6.2, we will pass an additional parameter $\mathsf{s} = \mathsf{s}(\lambda)$ to the $\mathsf{NfaABE.Setup}$, $\mathsf{NfaABE.Enc}$, $\mathsf{NfaABE.KeyGen}$ algorithms denoting the description size of NFAs that the scheme can deal with. Later we give a construction in Sec. 3.7 which can support NFAs with arbitrary size.

Definition 3.5.3 (Correctness)**.** An SKABE scheme NfaABE is correct if for all NFAs M, all $\mathbf{x} \in \Sigma^*$ such that $M(\mathbf{x}) = 1$ and for all messages $m \in \mathcal{M}$,

$$\Pr\left[\begin{array}{l} \mathsf{NfaABE.msk} \leftarrow \mathsf{NfaABE.Setup}(1^\lambda)\ , \\ \mathsf{NfaABE.sk}_M \leftarrow \mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk}, M)\ , \\ \mathsf{NfaABE.ct} \leftarrow \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \mathbf{x}, m)\ : \\ \mathsf{NfaABE.Dec}\big(\mathsf{NfaABE.sk}_M, M, \mathsf{NfaABE.ct}, \mathbf{x}\big) \neq \mathsf{m} \end{array}\right] = \mathrm{negl}(\lambda)$$

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

Definition 3.5.4 (Security for NfaABE)**.** The SKABE scheme NfaABE for a message space $\mathcal{M}$ is said to satisfy *selective security* if for any stateful PPT adversary A, there exists a negligible function $\mathrm{negl}(\cdot)$ such that

$$\mathsf{A}_{\mathsf{NfaABE,A}}(1^\lambda, \Sigma) \to \left|\Pr[\mathsf{Exp}^{(0)}_{\mathsf{NfaABE,A}}(1^\lambda) \to 1] - \Pr[\mathsf{Exp}^{(1)}_{\mathsf{NfaABE,A}}(1^\lambda) = 1]\right| \le \mathrm{negl}(\lambda),$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, the experiment $\mathsf{Exp}^{(b)}_{\mathsf{NfaABE},A}$, modeled as a game between the adversary A and a challenger, is defined as follows:

1. **Setup phase:** *At the beginning of the game,* A *takes as input* 1^λ *and declares its target* $X \subset \Sigma^*$*, which is a finite set of strings of arbitrary size. Then the challenger samples* $\mathsf{NfaABE.msk} \leftarrow \mathsf{NfaABE.Setup}(1^\lambda)$.
2. **Query phase:** *During the game,* A *adaptively makes the following queries, in an arbitrary order and unbounded many times.*
   (a) **Encryption queries:** A *submits to the challenger an attribute* $\mathbf{x} \in X$ *and a pair of messages* $(m^{(0)}, m^{(1)}) \in (\mathcal{M}_\lambda)^2$*. Then, the challenger replies with* $\mathsf{NfaABE.ct} \leftarrow \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \mathbf{x}, \mathsf{m}^{(\mathsf{b})})$ *in order.*
   (b) **Key queries:** A *submits to the challenger an NFA* M *such that* $M(\mathbf{x}) = 0$ *for all* $\mathbf{x} \in X$*. Then, the challenger replies with* $\mathsf{NfaABE.sk}_M \leftarrow \mathsf{NfaABE.KeyGen}$ $(\mathsf{NfaABE.msk}, M)$ *in order.*
3. **Output phase:** *A outputs a guess bit b' as the output of the experiment.*

Remark 3.5.5. As noted in Remark 3.5.2, our construction in Sec. 3.6.2 is indexed with an additional parameter s that specifies the size of NFAs being dealt with. In that case, the above security definitions are modified so that A chooses 1^{s} in addition to X (or X and M, in the case of very selective security) at the beginning of the game and key generation queries are made only for machines with size s.

3.5.3 Definitions: Attribute Based Encryption and Functional Encryption for circuits

Attribute based Encryption for Circuits

For $\lambda \in \mathbb{N}$, let $\mathcal{C}_{\mathsf{inp},\mathsf{d}}$ denote a family of circuits with inp bit inputs, an a-priori bounded depth d, and binary output and $\mathcal{C} = \{\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}\}_{\lambda\in\mathbb{N}}$. An attribute-based encryption (ABE) scheme ABE for $\mathcal{C}$ over a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of four algorithms:

- $\mathsf{ABE.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}})$ is a PPT algorithm takes as input the unary representation of the security parameter, the length $\mathsf{inp} = \mathsf{inp}(\lambda)$ of the input and the depth $\mathsf{d} = \mathsf{d}(\lambda)$ of the circuit family $\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}$ to be supported. It outputs the master public key and the master secret key $(\mathsf{ABE.mpk}, \mathsf{ABE.msk})$.
- $\mathsf{ABE.Enc}(\mathsf{ABE.mpk}, \mathbf{x}, m)$ is a PPT algorithm that takes as input the master public key $\mathsf{ABE.mpk}$, a string $\mathbf{x} \in \{0,1\}^{\mathsf{inp}}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext $\mathsf{ABE.ct}$.
- $\mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, C)$ is a PPT algorithm that takes as input the master secret key $\mathsf{ABE.msk}$ and a circuit $C \in \mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}$ and outputs a corresponding secret key $\mathsf{ABE.sk}_C$.

- $\mathsf{ABE.Dec}(\mathsf{ABE.mpk}, \mathsf{ABE.sk}_C, C, \mathsf{ABE.ct}, \mathbf{x})$ is a deterministic algorithm that takes as input the secret key $\mathsf{ABE.sk}_C$, its associated circuit C, a ciphertext $\mathsf{ABE.ct}$, and its associated string $\mathbf{x}$ and outputs either a message m' or $\perp$.

Definition 3.5.6 (Correctness)**.** An ABE scheme for circuits ABE is correct if for all $\lambda \in \mathbb{N}$, polynomially bounded inp and d, all circuits $C \in \mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}$, all $\mathbf{x} \in \{0,1\}^{\mathsf{inp}}$ such that $C(\mathbf{x}) = 1$ and for all messages $m \in \mathcal{M}$,

$$\Pr\left[\begin{array}{l} (\mathsf{ABE.mpk}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}), \\ \mathsf{ABE.sk}_C \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, C), \\ \mathsf{ABE.ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, \mathbf{x}, m) : \\ \mathsf{ABE.Dec}\Big(\mathsf{ABE.mpk}, \mathsf{ABE.sk}_C, C, \mathsf{ABE.ct}, \mathbf{x}\Big) \neq m \end{array}\right] = \mathrm{negl}(\lambda)$$

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

Definition 3.5.7 (Selective Security for ABE)**.** The ABE scheme ABE for a circuit family $\mathcal{C} = \{\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}\}_{\lambda\in\mathbb{N}}$ and a message space $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to satisfy *selective security* if for any stateful PPT adversary A, there exists a negligible function $\mathrm{negl}(\cdot)$ such that

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

for all sufficiently large $\lambda \in \mathbb{N}$, where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, the experiment $\mathsf{Exp}^{(b)}_{\mathsf{ABE},\mathsf{A}}$, modeled as a game between adversary A and a challenger, is defined as follows:

1. **Setup phase:** *On input* $1^\lambda, \mathsf{A}$ *submits* $(1^{\mathsf{inp}}, 1^{\mathsf{d}})$ *and the target* $X \subset \{0,1\}^{\mathsf{inp}}$*, which is a set of binary strings of length* inp*, to the challenger. The challenger samples* $(\mathsf{ABE.mpk}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}})$ *and replies to* A *with* $\mathsf{ABE.mpk}$.

2. **Query phase:** *During the game,* A *adaptively makes the following queries, in an arbitrary order and unbounded many times.*

   (a) **Key Queries:** A *chooses a circuit* $C \in \mathcal{C}_{\mathsf{inp},\mathsf{d}}$ *that satisfies* $C(\mathbf{x}) = 0$ *for all* $\mathbf{x} \in X$*. For each such query, the challenger replies with* $\mathsf{ABE.sk}_C \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, C)$.

   (b) **Encryption Queries:** A *submits a string* $\mathbf{x} \in X$ *and a pair of equal length messages* $(m_0, m_1) \in (\mathcal{M})^2$ *to the challenger. The challenger replies to* A *with* $\mathsf{ABE.ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, \mathbf{x}, \mathsf{m_b})$.

3. **Output phase:** *A outputs a guess bit* b' *as the output of the experiment.*

Remark 3.5.8. The above definition allows an adversary to make encryption queries multiple times. More standard notion of the security for an ABE restricts the adversary to make only a single encryption query. It is well-known that they are actually equivalent, which is shown by a simple hybrid argument. We adopt the above definition since it is convenient for our purpose.

In our construction of SKABE for NFA in Sec. 3.6.2, we will use the ABE scheme by Boneh et al. [Boneh *et al.* (2014)] as a building block. The following theorem summarizes the efficiency properties of their construction.

Theorem 3.5.9 (Adapted from [Boneh *et al.* (2014)]). *There exists a selectively secure ABE scheme* $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ *with the following properties under the LWE assumption.*

1. *The circuit* $\mathsf{ABE.Setup}(\cdot,\cdot,\cdot;\cdot)$*, which takes as input* $1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}$*, and a randomness* r *and outputs* $\mathsf{ABE.msk} = \mathsf{ABE.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}; r)$*, can be implemented with depth* $\mathrm{poly}(\lambda, \mathsf{d})$*. In particular, the depth of the circuit is independent of* inp *and the length of the randomness* r*.*
2. *We have* $|\mathsf{ABE.sk}_C| \leq \mathrm{poly}(\lambda, \mathsf{d})$ *for any* $C \in \mathcal{C}_{\mathsf{inp},\mathsf{d}}$*, where* $(\mathsf{ABE.mpk}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}})$ *and* $\mathsf{ABE.sk}_C \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, C)$*. In particular, the length of the secret key is independent of the input length* inp *and the size of the circuit* C*.*
3. *Let* $C : \{0,1\}^{\mathsf{inp}+\ell} \to \{0,1\}$ *be a circuit such that we have* $C[v] \in \mathcal{C}_{\mathsf{inp},d}$ *for any* $v \in \{0,1\}^\ell$*. Then, the circuit* $\mathsf{ABE.KeyGen}(\cdot,\cdot,C[\cdot];\cdot)$*, that takes as input* $\mathsf{ABE.mpk}$*,* $\mathsf{ABE.msk}$*,* v*, and randomness* $\widehat{\mathsf{R}}$ *and outputs* $\mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, C[v]; \widehat{\mathsf{R}})$*, can be implemented with depth* $\mathsf{depth}(C) \cdot \mathrm{poly}(\lambda, \mathsf{d})$*.*

Proof. We show that the construction proposed by Boneh et al. [Boneh *et al.* (2014)] satisfies the properties. We only focus on the third item of the theorem, as the first one is easy to observe and the second one is explicitly mentioned in the paper.

To give the proof, we briefly recall the setup and key generation algorithms of their scheme. The setup algorithm prepares a set of matrices $(\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_{\mathsf{inp}})$ and a vector $\mathbf{u}$, whose sizes only depend on λ and d. To generate a secret key for a circuit C (without a hardwired value), the key generation algorithm homomorphically evaluates the circuit on matrices $(\mathbf{A}_1, \ldots, \mathbf{A}_{\mathsf{inp}})$ in a gate by gate manner. In more details, it assigns $\mathbf{A}_i$ to the wire corresponding to the i-th bit of the input and computes a matrix for each internal wire of the circuit. The size of the matrices will be the same for all wires. In more details, let g be a gate with incoming wires w_1 and w_2 and output wire w_3. Then, the matrix corresponding to w_3 is computed from the matrices corresponding to w_1 and w_2, where the computation applied to the matrices depends on the type of the gate g. In the end, it obtains the matrix $\mathbf{A}_C$ corresponding to the output wire. Then, it generates a short vector $\mathbf{e}$ such that $[\mathbf{A} \| \mathbf{A}_C]\mathbf{e} = \mathbf{u}$ using the trapdoor for $\mathbf{A}$ and outputs $\mathbf{e}$ as a secret key.

We first show that in the case of $\ell = 0$, or equivalently in the case where a circuit $C : \{0,1\}^{\mathsf{inp}} \to \{0,1\}$ is not hardwired any value, the statement holds. To see this, we first observe that the last operation in which short vector $\mathbf{e}$ is sampled can be implemented by a circuit with size $\mathrm{poly}(\lambda, \mathsf{d})$, since the sizes of $\mathbf{A}$, $\mathbf{A}_C$, and $\mathbf{u}$ are bounded by $\mathrm{poly}(\lambda, \mathsf{d})$. We then focus on the computational cost of homomorphic operation on matrices. We can implement the circuit that performs this step with depth $\mathsf{depth}(C) \cdot \mathrm{poly}(\lambda, \mathsf{d})$ by just replacing each gate of the circuit C with a circuit that performs the homomorphic matrix operation corresponding to this gate.

We then consider the general case where $\ell \neq 0$. In this case, we first construct a circuit that performs homomorphic operations given matrices $\mathbf{B}_1, \ldots, \mathbf{B}_\ell, \mathbf{A}_1, \ldots, \mathbf{A}_{\mathsf{inp}}$ and $C(\cdot, \cdot)$, where $\mathbf{B}_1, \ldots, \mathbf{B}_\ell$ will correspond to the hardwired value. By the above discussion, such a circuit can be implemented with depth $\mathsf{depth}(C) \cdot \mathrm{poly}(\lambda, \mathsf{d})$. It remains to show that it is possible to construct a circuit that takes as input $\mathbf{A}_1, \ldots, \mathbf{A}_{\mathsf{inp}}$ and the hardwired value v and outputs matrices $\mathbf{B}_1, \ldots, \mathbf{B}_\ell$. Such a circuit can be implemented with depth $\mathrm{poly}(\lambda, \mathsf{d})$ by computing $\mathbf{B}^{(0)}$ and $\mathbf{B}^{(1)}$ that correspond to $0 = x_1 \wedge (\neg x_1)$ and $1 = x_1 \vee (\neg x_1)$ from $\mathbf{A}_1$ and then outputting $\mathbf{B}^{(v_1)}, \cdots, \mathbf{B}^{(v_\ell)}$, where v_i is the i-th bit of v. This completes the proof of the theorem. □

Remark 3.5.10. As we mentioned, we use ABE for circuits with the above efficiency properties to construct ABE for NFA in Sec. 3.6.2. Since we only have selectively secure ABE scheme satisfying the above properties, the resulting construction of ABE for NFA will only have selective security. One could consider that by applying the standard complexity leveraging argument to the selectively secure ABE scheme by Boneh et al. [Boneh *et al.* (2014)] to obtain an adaptively secure scheme and then using the resultant scheme in the construction in Sec. 3.6.2, we can obtain adaptively secure ABE scheme for NFA. This is not true because the resulting ABE scheme obtained by the complexity leveraging have secret keys whose size is polynomially dependent on the input length $\mathsf{inp}(\lambda)$ of the circuits and does not satisfy the second efficiency property in Theorem 3.5.9, which is crucial for the construction in Sec. 3.6.2 to work.

Functional Encryption for Circuits

For $\lambda \in \mathbb{N}$, let $\mathcal{C}_{\mathsf{inp},\mathsf{d},\mathsf{out}}$ denote a family of circuits with inp bit inputs, depth d, and output length out and $\mathcal{C} = \{\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda),\mathsf{out}(\lambda)}\}_{\lambda\in\mathbb{N}}$. A functional encryption (FE) scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for $\mathcal{C}$ consists of four algorithms:

- $\mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}, 1^{\mathsf{out}})$ is a PPT algorithm takes as input the unary representation of the security parameter, the length $\mathsf{inp} = \mathsf{inp}(\lambda)$ of the input, depth $\mathsf{d} = \mathsf{d}(\lambda)$, and the length of the output out $= \mathsf{out}(\lambda)$ of the circuit family $\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda),\mathsf{out}(\lambda)}$ to be supported. It outputs the master public key $\mathsf{FE.mpk}$ and the master secret key $\mathsf{FE.msk}$.

- $\mathsf{FE.KeyGen}(\mathsf{FE.mpk}, \mathsf{FE.msk}, C)$ is a PPT algorithm that takes as input the master public key $\mathsf{FE.mpk}$, master secret key $\mathsf{FE.msk}$, and a circuit $C \in \mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda),\mathsf{out}(\lambda)}$ and outputs a corresponding secret key $\mathsf{FE.sk}_C$.

- $\mathsf{FE.Enc}(\mathsf{FE.mpk}, \mathbf{x})$ is a PPT algorithm that takes as input the master public key $\mathsf{FE.mpk}$ and an input message $\mathbf{x} \in \{0,1\}^{\mathsf{inp}(\lambda)}$ and outputs a ciphertext $\mathsf{FE.ct}$.

- $\mathsf{FE.Dec}(\mathsf{FE.mpk}, \mathsf{FE.sk}_C, C, \mathsf{FE.ct})$ is a deterministic algorithm that takes as input the master public key $\mathsf{FE.mpk}$, a secret key $\mathsf{FE.sk}_C$, corresponding circuit C, and a ciphertext $\mathsf{FE.ct}$ and outputs $C(\mathbf{x})$.

Definition 3.5.11 (Correctness)**.** A functional encryption scheme CktFE is correct if for all $C \in \mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda),\mathsf{out}(\lambda)}$ and all $\mathbf{x} \in \{0,1\}^{\mathsf{inp}(\lambda)}$,

$$\Pr\left[\begin{array}{l}(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{d}(\lambda)}, 1^{\mathsf{out}(\lambda)});\\ \mathsf{FE.Dec}\Big(\mathsf{FE.mpk}, \mathsf{FE.KeyGen}(\mathsf{FE.mpk}, \mathsf{FE.msk}, C), C, \mathsf{FE.Enc}(\mathsf{FE.mpk}, \mathbf{x})\Big) \neq \mathsf{C}(\mathbf{x})\end{array}\right] = \mathrm{negl}(\lambda)$$

where the probability is taken over the coins of $\mathsf{FE.Setup}$, $\mathsf{FE.KeyGen}$, $\mathsf{FE.Enc}$ and, $\mathsf{FE.Dec}$).

We then define full simulation based security for single key FE as in [(Goldwasser *et al.*, 2013*a*, Defn 2.13)].

Definition 3.5.12 ($\mathsf{FULL\text{-}SIM}$ Security)**.** Let CktFE be a functional encryption scheme for a circuits. For a stateful PPT adversary A and a *stateless* PPT simulator Sim, consider the following two experiments:

```
Exp^{real}_{CktFE,A}(1^λ):
1: Let $\overrightarrow{\mathbf{v}} := (1^{\mathsf{inp}}, 1^{\mathsf{d}}, 1^{\mathsf{out}}) \leftarrow \mathsf{A}(1^\lambda)$
2: $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, \overrightarrow{\mathbf{v}})$
3: $C \leftarrow \mathsf{A}(\mathsf{FE.mpk})$
4: $\mathsf{FE.sk}_C \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}, \mathsf{FE.msk}, C)$
5: $\alpha \leftarrow \mathsf{A}^{\mathsf{FE.Enc}(\mathsf{FE.mpk},\cdot)}(\mathsf{FE.mpk}, \mathsf{FE.sk}_C)$

Exp^{ideal}_{CktFE,Sim}(1^λ):
1: Let $\overrightarrow{\mathbf{v}} := (1^{\mathsf{inp}}, 1^{\mathsf{d}}, 1^{\mathsf{out}}) \leftarrow \mathsf{A}(1^\lambda)$
2: $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, \overrightarrow{\mathbf{v}})$
3: $C \leftarrow \mathsf{A}(\mathsf{FE.mpk})$
4: $\mathsf{FE.sk}_C \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}, \mathsf{FE.msk}, C)$
5: $\alpha \leftarrow \mathsf{A}^{\mathsf{O}(\cdot)}(\mathsf{FE.mpk}, \mathsf{FE.sk}_C)$
```

Here, $\mathsf{O}(\cdot)$ is an oracle that on input $\mathbf{x}$ from A, runs Sim with inputs $(\mathsf{FE.mpk}, \mathsf{sk}_C, C, C(\mathbf{x}), 1^{\mathsf{inp}})$ to obtain a ciphertext $\mathsf{FE.ct}$ and returns it to the adversary A.

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

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

Remark 3.5.13. The above definition allows an adversary to make encryption queries multiple times. In the security notion defined in [Goldwasser *et al.* (2013*a*)], the adversary is allowed to make only a single encryption query. Similarly to the case of ABE, it is easy to see that these definitions are actually equivalent (See Remark 3.5.8). We adopt the above definition since it is convenient for our purpose.

Remark 3.5.14 (Selective Simulation Security.). We can consider a weaker version of the above security notion where A outputs a set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|X|}\} \subset \Sigma^*$ along with $(1^{\mathsf{inp}}, 1^{\mathsf{d}}, 1^{\mathsf{out}})$ at the beginning of the game and A is only allowed to query $\mathbf{x} \in X$ to $\mathsf{FE.Enc}(\mathsf{FE.mpk}, \cdot)$ and $\mathsf{O}(\cdot)$. We call this security notion selective simulation security.

In our construction of SKABE for NFA in Sec. 3.6.2, we will use the FE scheme by [Goldwasser *et al.* (2013*a*)] as a building block. The following theorem summarizes the efficiency properties of their construction.

Theorem 3.5.15 ([Goldwasser *et al.* (2013*a*)])**.** *There exists an FE scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *with the following properties.*

1. *For any polynomially bounded* $\mathsf{inp}(\lambda), \mathsf{d}(\lambda), \mathsf{out}(\lambda)$, *all the algorithms in* FE *run in polynomial time. Namely, the running time of* $\mathsf{FE.Setup}$ *and* $\mathsf{FE.Enc}$ *do not depend on the size of circuit description to be supported by the scheme.*

2. *Assuming the subexponential hardness of the LWE problem, the scheme satisfies full-simulation-based security.*

We note that the first property above is called succinctness or semi-compactness of FE. A stronger version of the efficiency property called compactness requires the running time of the encryption algorithm to be dependent only on the length of input message $\mathbf{x}$. An FE with compactness is known to imply indistinguishability obfuscation [Ananth and Jain (2015); Bitansky and Vaikuntanathan (2015)].

3.6 Attribute-based Encryption for NFA

3.6.1 NFA as NC circuit

Here, we introduce a theorem that provides an efficient algorithm that converts an NFA into an equivalent circuit with shallow depth. The shallowness of the circuit will play a crucial role in our construction of SKABE for NFA. In the following, for ease of notation, we often input a string in Σ^* to a circuit with the understanding that the input is actually a binary string encoding a string in Σ^*. To do so, we set $\eta := \lceil \log(|\Sigma| + 1) \rceil$ and regard a symbol in Σ as a binary string in $\{0,1\}^\eta$ by a natural injection map from Σ to $\{0,1\}^\eta$. Furthermore, we also introduce a special symbol $\perp$ that is not in Σ and assign an unused symbol in $\{0,1\}^\eta$ to it. Intuitively, $\perp$ represents a blank symbol that will be used to adjust the length of a string. We will use alphabets $\{0,1\}^\eta$ and $\Sigma \cup \{\perp\}$ interchangeably.

Theorem 3.6.1. *Let Σ be an alphabet for NFAs. Then we have the following:*

1. *There exists a family of circuits $\{\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}\}_{\mathsf{s},\ell \in \mathbb{N}}$ where the circuit $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}$ takes as input an NFA M with size s and outputs a circuit $\widehat{M_\ell} : (\Sigma \cup \{\perp\})^\ell \to \{0,1\}$. Furthermore, for all $\ell, \mathsf{s} \in \mathbb{N}$, all string $\mathbf{x} \in \Sigma^{\leq \ell}$, and all NFA M with size s, we have*
$$\widehat{M_\ell}(\hat{\mathbf{x}}) = M(\mathbf{x}),$$
where $\widehat{M_\ell} = \mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(M)$ and $\hat{\mathbf{x}} = \mathbf{x} \| \perp^{\ell - |\mathbf{x}|}$.

2. *The depths of the circuits $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}$ and $\widehat{M_\ell} = \mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(M)$ for an NFA M of size s are bounded by $\mathrm{poly}(\log \mathsf{s}, \log \ell)$. Furthermore, the sizes of these circuits are bounded by $\mathrm{poly}(\mathsf{s}, \ell)$.*

Proof. We define the circuit $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}$ as in Figure 3.1. There, we introduce a circuit M_{2^j} that takes as input $\mathbf{x} \in (\Sigma \cup \{\perp\})^{2^j}$ and outputs $\{b_{q,q',\mathbf{x}}\}_{(q,q') \in Q \times Q}$, where the boolean value $b_{q,q',\mathbf{x}} \in \{0,1\}$ is set to 1 if the state q' is reachable from q by reading $\mathbf{x}$ and 0 otherwise. Here, we augment the transition function T so that it works on the extended alphabet $\Sigma \cup \{\bot\}$, where we define $T(\bot, q) = \{q\}$ for all $q \in Q$. It is easily seen that the padding with $\bot$ and the augmentation of the transition function T we introduce here do not change the value of $M(\mathbf{x})$. We refer to Figure 3.2 for the concrete way of constructing M_{2^j}. It is not hard to see that $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}$ defined as in Figure 3.1 satisfies Item 1 of the theorem.

Circuit $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(M)$

1. Compute $b_{q,q',x}$ for all $(q,q',x) \in Q \times Q \times (\Sigma \cup \{\bot\})$ in parallel from M.
2. Then, construct the circuit M_1 from $\{b_{q,q',x}\}_{(q,q',x)\in Q\times Q\times(\Sigma\cup\{\bot\})}$, which takes $y \in (\Sigma \cup \{\bot\})$ as input, checks whether $y{=}x$ for all $x \in (\Sigma \cup \{\bot\})$ in parallel, and outputs $\{b_{q,q',x}\}_{(q,q')\in Q\times Q}$ such that $x = y$.
3. Compute M_{2^j} for $j \in [i]$ in the ascending order, where M_{2^j} is constructed from $M_{2^{j-1}}$ as in Figure 3.2 and $i = \lceil \log \ell \rceil$.
4. Compute $\widehat{M_\ell}$ defined as in Figure 3.3 from M_{2^i} and output $\widehat{M_\ell}$.

Figure 3.1: The Circuit $\mathsf{To\text{-}Circuit}$.

Circuit $M_{2^j}(\mathbf{x})$

1. Parse the input $\mathbf{x} = \mathbf{x}_0 \| \mathbf{x}_1$, where $\mathbf{x}_0, \mathbf{x}_1 \in (\Sigma \cup \{\bot\})^{2^{j-1}}$.
2. Compute $M_{2^{j-1}}(\mathbf{x}_0) = \{b_{q,q',\mathbf{x}_0}\}_{(q,q')\in Q\times Q}$ and $M_{2^{j-1}}(\mathbf{x}_1) = \{b_{q,q',\mathbf{x}_1}\}_{(q,q')\in Q\times Q}$ in parallel.
3. Compute $b_{q,q',\mathbf{x}}$ for all $(q,q') \in Q \times Q$ in parallel by executing the following:
   (a) Compute $(b_{q,q'',\mathbf{x}_0} \wedge b_{q'',q',\mathbf{x}_1})$ for all $q'' \in Q$ in parallel.
   (b) Compute $b_{q,q',\mathbf{x}} := \vee_{q''\in Q}(b_{q,q'',\mathbf{x}_0} \wedge b_{q'',q',\mathbf{x}_1})$.
4. Output $\{b_{q,q',\mathbf{x}}\}_{(q,q')\in Q\times Q}$.

Figure 3.2: The Circuit $M_{2^j}(\mathbf{x})$.

To finish the proof, we have to show Item 2 of the theorem. We first bound the size of $\widehat{M_\ell}$. To do this, we first observe that $\mathsf{size}(M_1) \le \mathrm{poly}(|\Sigma|, |Q|)$ holds. Furthermore, we have

$$\mathsf{size}(M_{2^j}) \le 2 \cdot \mathsf{size}(M_{2^{j-1}}) + \mathrm{poly}(|\Sigma|, |Q|) \quad \text{and} \quad \mathsf{depth}(\widehat{M_\ell}) \le \mathsf{depth}(M_{2^i}) + \mathrm{poly}(|\Sigma|, |Q|).$$

```
Circuit $\widehat{M_\ell}(\hat{\mathbf{x}})$

1. Pad the input $\hat{\mathbf{x}} \in (\Sigma \cup \{\perp\})^\ell$ to obtain $\tilde{\mathbf{x}} := \hat{\mathbf{x}} \| \perp^{2^i - \ell} \in (\Sigma \cup \{\perp\})^{2^i}$.
2. Compute $M_{2^i}(\tilde{\mathbf{x}}) = \{b_{q,q',\tilde{\mathbf{x}}}\}_{(q,q') \in Q \times Q}$.
3. Compute $b = \vee_{q \in F} b_{q_{\mathsf{st}},q,\tilde{\mathbf{x}}}$ and output $b$.
```

Figure 3.3: The Circuit $\widehat{M_\ell}(\hat{\mathbf{x}})$.

From the above, we have

$$\mathsf{size}(\widehat{M_\ell}) \leq 2^i \operatorname{poly}(|\Sigma|, |Q|) \leq \operatorname{poly}(\mathsf{s}, \ell) \tag{3.1}$$

as desired. We then bound the depth of $\widehat{M_\ell}$. We first observe $\mathsf{depth}(M_1) = \operatorname{poly}(\log|\Sigma|, \log|Q|)$. Furthermore, we have

$$\mathsf{depth}(M_{2^j}) \leq \mathsf{depth}(M_{2^{j-1}}) + \operatorname{poly}(\log|\Sigma|, \log|Q|)$$

and

$$\mathsf{depth}(\widehat{M_\ell}) \leq \mathsf{depth}(M_{2^i}) + \operatorname{poly}(\log|\Sigma|, \log|Q|).$$

From the above, we have

$$\mathsf{depth}(\widehat{M_\ell}) \leq i \cdot \operatorname{poly}(\log|\Sigma|, \log|Q|) \leq \operatorname{poly}(\log \mathsf{s}, \log \ell)$$

as desired.

We next bound the size of the circuit $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(\cdot)$. It is easy to see that Step 1 and 2 of $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(\cdot)$ can be implemented by circuits of size $\operatorname{poly}(|\Sigma|, |Q|)$. We also observe that j-th repetition in Step 3 can be implemented by a circuit of size $\operatorname{poly}(\mathsf{size}(M_{2^{j-1}}), |\Sigma|, |Q|) \leq \operatorname{poly}(\mathsf{size}(\widehat{M_\ell}), |\Sigma|, |Q|)$. We can also see that Step 4 can be implemented by a circuit of size $\operatorname{poly}(\mathsf{size}(\widehat{M_\ell}), |\Sigma|, |Q|)$. Therefore, we have

$$\mathsf{size}(\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}) \leq i \cdot \operatorname{poly}(\mathsf{size}(\widehat{M_\ell}), |\Sigma|, |Q|) \leq \operatorname{poly}(\mathsf{s}, \ell)$$

as desired, where the second inequality follows from Eq. (3.1).

We finally bound the depth of the circuit $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(\cdot)$. It is easy to see that Step 1, 2, and 4 of $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(\cdot)$ can be implemented by circuits of depth $\mathrm{poly}(\log|\Sigma|, \log|Q|)$. We also observe that each repetition in Step 3 can be implemented with depth $\mathrm{poly}(\log|\Sigma|, \log|Q|)$, since it just copies $M_{2^{j-1}}$ and adds a fixed circuit to it that performs Item 3 and 4 of M_{2^j}. Therefore, Step 3 of $\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}(\cdot)$ can be implemented by a circuit of depth $i \cdot \mathrm{poly}(\log|\Sigma|, \log|Q|)$. To sum up, we have

$$\mathsf{depth}(\mathsf{To\text{-}Circuit}_{\mathsf{s},\ell}) \le i \cdot \mathrm{poly}(\log|\Sigma|, \log|Q|) \le \mathrm{poly}(\log \mathsf{s}, \log \ell)$$

as desired. This completes the proof of the theorem. □

3.6.2 Construction: SKABE for Bounded Size NFA

We construct an SKABE scheme for NFA denoted by $\mathsf{NfaABE} = (\mathsf{NfaABE.Setup}, \mathsf{NfaABE.KeyGen}, \mathsf{NfaABE.Enc}, \mathsf{NfaABE.Dec})$ from the following ingredients:

1. $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: a pseudorandom function, where a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ defines a function $\mathsf{PRF.Eval}(\mathsf{K}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$. We denote the length of K by $|\mathsf{K}|$.

2. $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$: a functional encryption scheme for circuit with the efficiency property described in Item 1 of Theorem 3.5.15. We can instantiate FE with the scheme proposed by [Goldwasser *et al.* (2013*a*)].

3. $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$: An ABE scheme that satisfies the efficiency properties described in Theorem 3.5.9. We can instantiate ABE with the scheme proposed by [Boneh *et al.* (2014)].

4. $U(\cdot,\cdot)$: a universal circuit that takes as input a circuit C of fixed depth and size and an input $\mathbf{x}$ to the circuit and outputs $C(\mathbf{x})$. We often denote by $U[C](\cdot) = U(C, \cdot)$ a universal circuit U with the first input C being hardwired. We need to have $\mathsf{depth}(U) \le O(\mathsf{depth}(C))$. For construction of such a universal circuit, we refer to [Cook and Hoover, (1985)].

Below we provide our construction for SKABE for NFA. In the description below, we abuse notation and denote as if the randomness used in a PPT algorithm was a key K of the pseudorandom function PRF. Namely, for a PPT algorithm (or circuit) A that takes as input x and a randomness $r \in \{0,1\}^\ell$ and outputs y, $\mathsf{A}(x; \mathsf{K})$ denotes an algorithm that computes $r := \mathsf{PRF}(\mathsf{K}, 1) \| \mathsf{PRF}(\mathsf{K}, 2) \| \cdots \| \mathsf{PRF}(\mathsf{K}, \ell)$ and runs $\mathsf{A}(x; r)$. Note that if A is a circuit, this transformation makes the size of the circuit polynomially larger and adds a fixed polynomial overhead to its depth. In particular, even if we add this change to $\mathsf{ABE.Setup}$ and $\mathsf{ABE.KeyGen}$, the efficiency properties of ABE described in Theorem 3.5.9 are preserved.

$\mathsf{NfaABE.Setup}(1^\lambda, 1^{\mathsf{s}})$: On input the security parameter 1^λ and a description size s of an NFA, do the following:

1. For all $j \in [0, \lambda]$, sample PRF keys $\widehat{\mathsf{K}}_j, \mathsf{R}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
2. For all $j \in [0, \lambda]$, sample $(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{out}(\lambda)}, 1^{\mathsf{d}(\lambda)})$.

   Here, we generate $\lambda + 1$ instances of FE. Note that all instances support a circuit class with input length $\mathsf{inp}(\lambda) = \mathsf{s} + 2|\mathsf{K}|$, output length $\mathsf{out}(\lambda)$, and depth $\mathsf{d}(\lambda)$, where $\mathsf{out}(\lambda)$ and $\mathsf{d}(\lambda)$ are polynomials in the security parameter that will be specified later.
3. Output $\mathsf{NfaABE.msk} = (\{\widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{FE.mpk}_j, \mathsf{FE.msk}_j\}_{j\in[0,\lambda]})$.

$\mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \mathbf{x}, m, 1^{\mathsf{s}})$: On input the master secret key $\mathsf{NfaABE.msk}$, an attribute $\mathbf{x} \in \Sigma^*$ of length at most 2^λ, a message m and the description size s of NFA, do the following:

1. Parse the master secret key as $\mathsf{NfaABE.msk} \to (\{\widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{FE.mpk}_j, \mathsf{FE.msk}_j\}_{j\in[0,\lambda]})$.
2. Set $\hat{\mathbf{x}} = \mathbf{x} \| \perp^{2^i - \ell}$, where $\ell = |\mathbf{x}|$ and $i = \lceil \log \ell \rceil$.
3. Compute an ABE key pair $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) = \mathsf{ABE.Setup}(1^\lambda, 1^{2^i \eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}}_i)$ with $\widehat{\mathsf{K}}_i$ as the randomness.

   Here, we generate an instance of ABE that supports a circuit class with input domain $\{0,1\}^{2^i\eta} \supseteq (\Sigma \cup \{\perp\})^{2^i}$ and depth $\hat{\mathsf{d}}$.
4. Compute $\mathsf{ABE.ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk_i}, \hat{\mathbf{x}}, \mathsf{m})$ as an ABE ciphertext for the message m under attribute $\hat{\mathbf{x}}$.
5. Obtain $\mathsf{FE.sk}_i = \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i, C_{\mathsf{s},2^i}; \mathsf{R}_i)$, where $C_{\mathsf{s},2^i}$ is a circuit described in Figure 3.4.
6. Output $\mathsf{NfaABE.ct} = (\mathsf{FE.sk_i}, \mathsf{ABE.mpk_i}, \mathsf{ABE.ct})$.

$\mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk}, M, 1^{\mathsf{s}})$: On input the master secret key $\mathsf{NfaABE.msk}$, the description of an NFA M and a size s of the NFA, if $|M| \neq \mathsf{s}$, output $\perp$ and abort. Else, proceed as follows.

1. Parse the master secret key as $\mathsf{NfaABE.msk} \to (\{\widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{FE.mpk}_j, \mathsf{FE.msk}_j\}_{j\in[0,\lambda]})$.
2. Sample $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for all $j \in [0, \lambda]$.
3. Compute $\mathsf{FE.ct}_j = \mathsf{FE.Enc}(\mathsf{FE.mpk}_j, (M, \widehat{\mathsf{K}}_j, \widehat{\mathsf{R}}_j))$ for all $j \in [0, \lambda]$.
4. Output $\mathsf{NfaABE.sk}_M = \{\mathsf{FE.ct}_j\}_{j\in[0,\lambda]}$.

Function $C_{\mathsf{s},2^i}$

1. Parse the input $\mathbf{w} = (M, \widehat{\mathsf{K}}, \widehat{\mathsf{R}})$, where M is an NFA and $\widehat{\mathsf{K}}$ and $\widehat{\mathsf{R}}$ are PRF keys.
2. Compute $(\mathsf{ABE.mpk}, \mathsf{ABE.msk}) = \mathsf{ABE.Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}})$.
3. Compute $\widehat{M_{2^i}} = \mathsf{To\text{-}Circuit}_{\mathsf{s},2^i}(M)$. (See Theorem 3.6.1 for the definition of $\mathsf{To\text{-}Circuit}$.)
4. Compute and output $\mathsf{ABE.sk}_{U[\widehat{M_{2^i}}]} = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, U[\widehat{M_{2^i}}]; \widehat{\mathsf{R}})$.

Figure 3.4: Circuit $C_{\mathsf{s},2^i}$, supported by the FE scheme. $C_{\mathsf{s},2^i}$ takes NFA M as input and outputs a secret key for the universal circuit $U[\widehat{M_{2^i}}]$ (hardwired with $\widehat{M_{2^i}}$) under the ABE scheme.

$\mathsf{NfaABE.Dec}(\mathsf{NfaABE.sk}_M, M, \mathsf{NfaABE.ct}, \mathbf{x})$: On input a secret key for NFA M and a ciphertext encoded under attribute $\mathbf{x}$, proceed as follows:

1. Parse the secret key as $\mathsf{NfaABE.sk}_M \to \{\mathsf{FE.ct}_j\}_{j\in[0,\lambda]}$ and the ciphertext as $\mathsf{NfaABE.ct} \to (\mathsf{FE.sk_i}, \mathsf{ABE.mpk_i}, \mathsf{ABE.ct})$.
2. Set $\ell = |\mathbf{x}|$ and choose $\mathsf{FE.ct}_i$ from $\mathsf{NfaABE.sk}_M = \{\mathsf{FE.ct}_j\}_{j\in[0,\lambda]}$ such that $i = \lceil \log \ell \rceil < \lambda$.
3. Compute $y = \mathsf{FE.Dec}(\mathsf{FE.mpk}_i, \mathsf{FE.sk}_i, C_{\mathsf{s},2^i}, \mathsf{FE.ct}_i)$.
4. Compute and output $z = \mathsf{ABE.Dec}(\mathsf{ABE.mpk}_i, y, U[\widehat{M_{2^i}}], \mathsf{ABE.ct_i}, \hat{\mathbf{x}})$, where we interpret y as an ABE secret key and $\hat{\mathbf{x}} = \mathbf{x}||\bot^{2^i-\ell}$.

3.6.3 Correctness of NfaABE

The following theorem asserts that our scheme is efficient.

Theorem 3.6.2. *Let* $|\Sigma|$, $\mathsf{d}(\lambda)$, $\hat{\mathsf{d}}(\lambda)$, *and* $\mathsf{out}(\lambda)$, *be polynomials in* λ. *Then,* $\mathsf{NfaABE} = (\mathsf{NfaABE.Setup}, \mathsf{NfaABE.KeyGen}, \mathsf{NfaABE.Enc}, \mathsf{NfaABE.Dec})$ *defined above runs in polynomial time.*

Proof. It is easy to see that the $\mathsf{NfaABE.Setup}$ and $\mathsf{NfaABE.KeyGen}$ run in polynomial time.

We then show that $\mathsf{NfaABE.Enc}$ runs in polynomial time. By the efficiency of ABE, it suffices to show that $C_{\mathsf{s},2^i}$ can be computed in polynomial time. To see this, we bound the time for constructing each step of the circuit. Trivially, Step 1 can be implemented with no cost. We first observe that Step 2 of the circuit can be implemented by a circuit with size $\mathrm{poly}(\lambda, 2^i\eta, \hat{\mathsf{d}}) = \mathrm{poly}(\lambda, |\mathbf{x}|)$ by the efficiency of $\mathsf{ABE.KeyGen}$. We then observe that Step 3 of the circuit can be implemented with size $\mathrm{poly}(\mathsf{s}, 2^i) \le \mathrm{poly}(\mathsf{s}, |\mathbf{x}|)$ by Item 2 of Theorem 3.6.1. Finally, Step 4 of the circuit can be implemented with size $\mathrm{poly}(\lambda, 2^i\eta, \hat{\mathsf{d}}, \mathsf{s}) \le \mathrm{poly}(\lambda, \mathsf{s}, |\mathbf{x}|)$ by the efficiency of the universal circuit and ABE and Item 2 of Theorem 3.6.1.

We finally bound the running time of NfaABE.Dec. Trivially, Step 1 and 2 of NfaABE.Dec can be implemented with no cost. By the efficiency of FE, the running time of Step 3 is also bounded by $\mathrm{poly}(\lambda, \mathsf{s}, |\mathbf{x}|)$. Finally, to bound the running time of Step 4, it suffices to bound the time for constructing $U[\widehat{M}_{\mathsf{s},2^i}]$ by the efficiency of ABE. Since $\widehat{M}_{\mathsf{s},2^i}$ can be constructed in time $\mathrm{poly}(\mathsf{s}, 2^i) \le \mathrm{poly}(\mathsf{s}, |\mathbf{x}|)$ by Item 2 of Theorem 3.6.1, so is $U[\widehat{M}_{\mathsf{s},2^i}]$. This completes the proof of the theorem. □

The following theorem addresses the correctness of the scheme.

Theorem 3.6.3. *For appropriately chosen* $\hat{\mathsf{d}}(\lambda)$, $\mathsf{out}(\lambda)$*, and* $\mathsf{d}(\lambda)$*, our scheme* NfaABE *is correct for any polynomially bounded* $\mathsf{s}(\lambda)$.

Proof. We have to show that if we set $\hat{\mathsf{d}}(\lambda)$, $\mathsf{out}(\lambda)$, and $\mathsf{d}(\lambda)$ appropriately, we have $z = m$ when $M(\mathbf{x}) = 1$, where z is the value retrieved in Step 4 of the decryption algorithm. To show this, let us set $\hat{\mathsf{d}}(\lambda) = \Omega(\lambda)$ and assume that

$$y = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M}_{2^i}]; \widehat{\mathsf{R}}_i) \tag{3.2}$$

holds for the moment, where y is the value retrieved in Step 3 of the decryption algorithm. Then, we have $z = m$ by the correctness of ABE if $U[\widehat{M}_{2^i}]$ is supported by the scheme, since we have

$$U[\widehat{M}_{2^i}](\hat{\mathbf{x}}) = \widehat{M}_{2^i}(\hat{\mathbf{x}}) = M(\mathbf{x}) = 1$$

by Item 1 of Theorem 3.6.1. We claim that the depth of $U[\widehat{M}_{2^i}]$ is at most $\hat{\mathsf{d}}$ and therefore $U[\widehat{M}_{2^i}]$ is indeed supported by the scheme. To see this, we observe that

$$\begin{aligned}
\mathsf{depth}(U[\widehat{M}_{2^i}]) &\le \mathsf{depth}(U(\cdot,\cdot)) + O(1) \\
&\le O(1)\cdot \mathsf{depth}(\widehat{M}_{2^i}) + O(1) \\
&\le \mathrm{poly}(\log \mathsf{s}, \log 2^i) \\
&\le \mathrm{poly}(\log \lambda) \\
&\le \hat{\mathsf{d}}
\end{aligned} \tag{3.3}$$

holds, where the second inequality follows from the property of the depth preserving universal circuit U and the third from Item 2 of Theorem 3.6.1.

It remains to prove that Eq. (3.2) holds if we set $\mathsf{d}(\lambda)$ and $\mathsf{out}(\lambda)$ appropriately. To do so, we show that the depth and the output length of $C_{\mathsf{s},2^i}$ are bounded by some fixed polynomials. By taking $\mathsf{d}(\lambda)$ and $\mathsf{out}(\lambda)$ larger than these polynomials, we can ensure that the circuit $C_{\mathsf{s},2^i}$ is supported by the FE scheme and thus Eq. (3.2) follows from the correctness of the FE, since we have

$$C_{\mathsf{s},2^i}(M, \widehat{\mathsf{K}}_i, \widehat{\mathsf{R}}_i) = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M_{2^i}}]; \widehat{\mathsf{R}}_i),$$

where $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) = \mathsf{ABE.Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}}_i)$ by the definition of $C_{\mathsf{s},2^i}$. We first bound the depth of $C_{\mathsf{s},2^i}$. To do so, we first observe that Step 2 of $C_{\mathsf{s},2^i}$ can be implemented by a circuit of depth $\mathrm{poly}(\lambda, \hat{\mathsf{d}}) = \mathrm{poly}(\lambda)$ by Item 1 of Theorem 3.5.9. We then observe that Step 3 of $C_{\mathsf{s},2^i}$ can be implemented by a circuit of depth $\mathrm{poly}(\log \mathsf{s}, \log 2^i) = \mathrm{poly}(\log \lambda)$ by Item 2 of Theorem 3.6.1. We then bound the depth of the circuit that implements Step 4 of $C_{\mathsf{s},2^i}$. This step is implemented by the circuit $\mathsf{ABE.KeyGen}(\cdot, \cdot, U[\cdot]; \cdot)$ that takes as input $\mathsf{ABE.mpk}_i$, $\mathsf{ABE.msk}_i$, $U[\widehat{M_{2^i}}]$ constructed in the previous step, and $\widehat{\mathsf{R}}$ and returns $\mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M_{2^i}}]; \widehat{\mathsf{R}})$. We have

$$\begin{aligned}\mathsf{depth}(\mathsf{ABE.KeyGen}(\cdot, \cdot, U[\cdot]; \cdot)) &\leq \mathrm{poly}(\lambda, \hat{\mathsf{d}}) \cdot \mathsf{depth}(U(\cdot, \cdot)) \\ &\leq \mathrm{poly}(\lambda, \hat{\mathsf{d}}) \cdot \hat{\mathsf{d}} \\ &\leq \mathrm{poly}(\lambda),\end{aligned}$$

where the first inequality follows from Item 3 of Theorem 3.5.9 and the second from Eq. (3.3). To sum up, we have that the depth of the circuit $C_{\mathsf{s},2^i}$ is bounded by some fixed polynomial.

We next bound the output length of $C_{\mathsf{s},2^i}$. Since the output of the circuit is $\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]} = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M_{2^i}}]; \widehat{\mathsf{R}})$, we bound the length of the ABE secret key. We have

$$|\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]}| \leq \mathrm{poly}(\lambda, \hat{\mathsf{d}}) \leq \mathrm{poly}(\lambda, \mathrm{poly}(\lambda)) \leq \mathrm{poly}(\lambda)$$

as desired, where the first inequality follows from the Item 2 of Theorem 3.5.9. This completes the proof of the theorem. □

3.6.4 Proof of Security for NfaABE

Here, we prove that NfaABE defined above is secure, if so are FE and ABE. Formally, we have the following theorem.

Theorem 3.6.4. *Assume that* FE *satisfies full simulation based security,* ABE *is selectively secure, and that* PRF *is a secure pseudorandom function. Then,* NfaABE *satisfies selective security.*

Proof. To prove the theorem, let us fix a PPT adversary A and introduce the following game $\mathbf{Game}_i$ between the challenger and A for $i \in [0, \lambda]$.

$\mathbf{Game}_i$**:** The game proceeds as follows.

Setup phase. At the beginning of the game, A takes 1^λ as input and submits 1^{s} and its target set $X \subset \Sigma^*$ to the challenger. Then, the challenger chooses $\mathsf{NfaABE.msk} \leftarrow \mathsf{NfaABE.Setup}(1^\lambda, 1^{\mathsf{s}})$.

The challenger answers the encryption and key queries made by A as follows.

Encryption queries. Given two messages $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, the challenger sets $\ell := |\mathbf{x}|$ and computes

$$\mathsf{NfaABE.ct} = \begin{cases} \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \hat{\mathbf{x}}, \mathsf{m}^{(0)}) & \text{If } \lceil \log \ell \rceil \geq i \\ \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \hat{\mathbf{x}}, \mathsf{m}^{(1)}) & \text{If } \lceil \log \ell \rceil \leq i - 1. \end{cases}$$

Then, it returns $\mathsf{NfaABE.ct}$ to A.

Key queries. Given an NFA M from A, the challenger runs $\mathsf{NfaABE.sk}_M \leftarrow \mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk}, M)$ and returns $\mathsf{NfaABE.sk}_M$ to A.

Finally, A outputs its guess b'.

In the following, let $\mathsf{E}_{\mathrm{xxx}}$ denote the probability that A outputs 1 in $\mathbf{Game}_{\mathrm{xxx}}$. It suffices to prove $|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_{\lambda+1}]| = \mathrm{negl}(\lambda)$, since $\mathbf{Game}_0$ (resp., $\mathbf{Game}_{\lambda+1}$) corresponds to the selective security game with $b = 0$ (resp., $b = 1$). Since we have

$$|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_{\lambda+1}]| \le \sum_{i\in[0,\lambda]} |\Pr[\mathsf{E}_i] - \Pr[\mathsf{E}_{i+1}]|$$

by the triangle inequality, it suffices to show $|\Pr[\mathsf{E}_i] - \Pr[\mathsf{E}_{i+1}]| = \mathrm{negl}(\lambda)$ for $i \in [0, \lambda]$. Let us define $\ell_{\max}$ and $i_{\max}$ as

$$\ell_{\max} := \max\{|\mathbf{x}| : \mathbf{x} \in X\} \qquad \text{and} \qquad i_{\max} := \lceil \log \ell_{\max} \rceil.$$

Note that $\ell_{\max}$ is bounded by the running time of A and thus is polynomial in λ. We then observe that for $i > i_{\max}$, we have $\mathbf{Game}_i = \mathbf{Game}_{\lambda+1}$ and thus $\Pr[\mathsf{E}_i] - \Pr[\mathsf{E}_{i+1}] = 0$. Therefore, in the following, we will show that $|\Pr[\mathsf{E}_i] - \Pr[\mathsf{E}_{i+1}]| = \mathrm{negl}(\lambda)$ holds for $i \le i_{\max}$. To do so, we further introduce the following sequence of games for $i \in [0, i_{\max}]$:

$\mathbf{Game}_{i,0}$: The game is the same as $\mathbf{Game}_i$.

$\mathbf{Game}_{i,1}$: In this game, we change the setup phase and the way encryption queries are answered as follows.

Setup phase. Given $X \subset \Sigma^*$ from A, the challenger chooses $\mathsf{NfaABE.msk} \leftarrow \mathsf{NfaABE.Setup}(1^\lambda, 1^{\mathsf{s}})$ as in the previous game. In addition, it computes

$$(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}}_i)$$

and

$$\mathsf{FE.sk}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i, C_{\mathsf{s},2^i}; \mathsf{R}_i).$$

Encryption queries. Given two messages $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, the challenger sets $\ell := |\mathbf{x}|$ and computes $\mathsf{NfaABE.ct}$ as in the previous game if $\lceil \log \ell \rceil \ne i$. Otherwise, it computes

$$\mathsf{ABE.ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk_i}, \hat{\mathbf{x}}, \mathsf{m}^{(0)})$$

and returns $\mathsf{NfaABE.ct} = (\mathsf{FE.sk_i}, \mathsf{ABE.mpk_i}, \mathsf{ABE.ct})$ to A, where $\mathsf{FE.sk}_i$ and $\mathsf{ABE.mpk}_i$ are the values that are computed in the setup phase.

$\mathbf{Game}_{i,2}$: In this game, the challenger samples $\mathsf{FE.sk}_i$ as

$$\mathsf{FE.sk}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i, C_{\mathsf{s},2^i})$$

in the setup phase. Namely, it is sampled using true randomness instead of the pseudorandom bits derived from the PRF key R_i.

$\mathbf{Game}_{i,3}$: We change the way key queries are answered as follows:

Key queries. Given an NFA M of size s from A, the challenger answers the query as follows. It first chooses $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for $j \in [0, \lambda]$ and computes

$$\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]} = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M}_{2^i}]; \widehat{\mathsf{R}}_i),$$

where $\mathsf{ABE.mpk}_i$ and $\mathsf{ABE.msk}_i$ are the values that are computed in the setup phase. It then computes

$$\mathsf{FE.ct_j} \leftarrow \begin{cases} \mathsf{FE.Enc}(\mathsf{FE.mpk_j}, (\mathsf{M}, \widehat{\mathsf{K}}_\mathsf{j}, \widehat{\mathsf{R}}_\mathsf{j})) & \text{If } j \in [0,\lambda]\backslash\{i\} \\ \mathrm{Sim}(\mathsf{FE.mpk}_i, \mathsf{FE.sk}_i, C_{\mathsf{s},2^i}, \mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]}, 1^{\mathsf{inp}(\lambda)}) & \text{If } j = i. \end{cases} \tag{3.4}$$

Then, it returns $\mathsf{NfaABE.sk}_M := \{\mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]}$ to A.

$\mathbf{Game}_{i,4}$: In this game, the challenger samples $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)$ in the setup phase as

$$(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}).$$

It also generates $\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]}$ as

$$\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M}_{2^i}]).$$

when answering a key query. Namely, they are sampled using true randomness instead of the pseudorandom bits derived from the PRF keys $\widehat{\mathsf{K}}_i$ and $\widehat{\mathsf{R}}_i$.

$\mathbf{Game}_{i,5}$: In this game, we change the way the encryption queries are answered as follows.

Encryption queries. Given two messages $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, the challenger sets $\ell := |\mathbf{x}|$ and computes $\mathsf{NfaABE.ct}$ as in the previous game if

$\lceil \log \ell \rceil \neq i$. Otherwise, it computes

$$\mathsf{ABE.ct} = \mathsf{ABE.Enc}(\mathsf{ABE.mpk_i}, \hat{\mathbf{x}}, \mathsf{m}^{(1)})$$

and returns $\mathsf{NfaABE.ct} = (\mathsf{FE.sk_i}, \mathsf{ABE.mpk_i}, \mathsf{ABE.ct})$ to A, where $\mathsf{FE.sk}_i$ and $\mathsf{ABE.mpk}_i$ are the values that are computed in the setup phase.

$\mathbf{Game}_{i,6}$**:** The game is the same as $\mathbf{Game}_{i+1}$.

Since we have

$$|\Pr[\mathsf{E}_i] - \Pr[\mathsf{E}_{i+1}]| \leq \sum_{j\in[6]} |\Pr[\mathsf{E}_{i,j-1}] - \Pr[\mathsf{E}_{i,j}]|$$

by the triangle inequality, it suffices to show $|\Pr[\mathsf{E}_{i,j-1}] - \Pr[\mathsf{E}_{i,j}]| = \mathrm{negl}(\lambda)$ for $j \in [6]$. To complete the proof of the theorem, it remains to prove the following lemmas.

Lemma 3.6.5. *We have* $\Pr[\mathsf{E}_{i,0}] = \Pr[\mathsf{E}_{i,1}]$.

Proof. The change introduced here is only conceptual, where $\mathsf{ABE.mpk}_i$ and $\mathsf{FE.sk}_i$ are computed beforehand. The lemma trivially follows. □

Lemma 3.6.6. *We have* $|\Pr[\mathsf{E}_{i,1}] - \Pr[\mathsf{E}_{i,2}]| = \mathrm{negl}(\lambda)$.

Proof. We observe that R_i is used only when generating $\mathsf{FE.sk}_i$ in $\mathbf{Game}_{i,1}$. Therefore, the lemma follows by a straightforward reduction to the security of PRF. □

Lemma 3.6.7. *We have* $|\Pr[\mathsf{E}_{i,2}] - \Pr[\mathsf{E}_{i,3}]| = \mathrm{negl}(\lambda)$.

Proof. To prove the lemma, let us assume that $|\Pr[\mathsf{E}_{i,2}] - \Pr[\mathsf{E}_{i,3}]|$ is non-negligible and construct an adversary B that breaks the full simulation security of FE using A. B proceeds as follows.

Setup phase. At the beginning of the game, B inputs 1^λ to A and obtains 1^{s} and $X \subset \Sigma^*$ from A. Then B submits its target $(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{out}(\lambda)})$. Then, the experiment samples

$$(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{out}(\lambda)})$$

and returns $\mathsf{FE.mpk}$ to B. B then sets $\mathsf{FE.mpk}_i := \mathsf{FE.mpk}$. In the rest of the simulation, it implicitly sets $\mathsf{FE.msk}_i := \mathsf{FE.msk}$ without knowing the value. B then chooses $(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{out}(\lambda)}, 1^{\mathsf{d}(\lambda)})$ for $j \in [0,\lambda]\backslash\{i\}$. It also chooses $\widehat{\mathsf{K}}_j, \mathsf{R}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for $j \in [0,\lambda]$ and $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}}_i)$. Finally, it declares $C_{\mathsf{s},2^i}$ as a circuit for which it request a secret key. Then, the experiment runs

$$\mathsf{FE.sk} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}, \mathsf{FE.msk}, C_{\mathsf{s},2^i})$$

and returns $\mathsf{FE.sk}$ to B. B sets $\mathsf{FE.sk}_i := \mathsf{FE.sk}$.

B then handles the encryption and key queries as follows.

Encryption queries. Given two messages $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, $\mathcal{B}$ sets $\ell := |\mathbf{x}|$ and $i' = \lceil \log \ell \rceil$. If $i' \neq i$, B answers the query using $(\widehat{\mathsf{K}}_{i'}, \mathsf{R}_{i'}, \mathsf{FE.mpk}_{i'}, \mathsf{FE.msk}_{i'})$. Otherwise, it computes $\mathsf{ABE.ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_i, \hat{\mathbf{x}}, m^{(0)})$ and returns $\mathsf{NfaABE.ct} = (\mathsf{FE.sk}_i, \mathsf{ABE.mpk}_i, \mathsf{ABE.ct})$ to A, where $\mathsf{ABE.mpk}_i$ (resp., $\mathsf{FE.sk}_i$) is the value sampled by itself (resp., by the experiment) in the setup phase.

Key queries. Given an NFA M of size s from A, B first chooses $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for $j \in [0,\lambda]$ and computes $\mathsf{FE.ct_j} = \mathsf{FE.Enc}(\mathsf{FE.mpk_j}, (\mathsf{M}, \widehat{\mathsf{K}}_\mathsf{j}, \widehat{\mathsf{R}}_\mathsf{j}))$ for $j \in [0,\lambda]\backslash\{i\}$. B then submits $(M, \widehat{\mathsf{K}}_i, \widehat{\mathsf{R}}_i)$ to its encryption oracle. Then, the experiment computes

$$\mathsf{FE.ct} \leftarrow \begin{cases} \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{M}, \widehat{\mathsf{K}}_\mathsf{i}, \widehat{\mathsf{R}}_\mathsf{i})) & \text{If } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{CktFE,B}}(1^\lambda) \\ \mathrm{Sim}(\mathsf{FE.mpk}, \mathsf{FE.sk}, C_{\mathsf{s},2^i}, C_{\mathsf{s},2^i}(M, \widehat{\mathsf{K}}_i, \widehat{\mathsf{R}}_i), 1^{\mathsf{inp}(\lambda)}) & \text{If } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CktFE},\mathrm{Sim}}(1^\lambda) \end{cases} \tag{3.5}$$

and returns $\mathsf{FE.ct}$ to B. B then sets $\mathsf{FE.ct_i} := \mathsf{FE.ct}$ and returns $\mathsf{NfaABE.sk}_M := \{\mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]}$ to A.

Output phase: B outputs the same bit as A as its guess.

It is easy to see that B simulates $\mathbf{Game}_{i,2}$ if B is in the real game. We then claim that B simulates $\mathbf{Game}_{i,3}$ if B is in the simulated game. The only difference between these games is the way $\mathsf{FE.ct_i}$ is computed. In $\mathbf{Game}_{i,3}$, it is generated as Eq. (3.4) while in the simulation above, it is generated as Eq. (3.5) (with B being in $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CktFE},\mathrm{Sim}}$). However, they are equivalent because $\mathcal{B}$ has set $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) := (\mathsf{FE.mpk}, \mathsf{FE.msk})$ and $\mathsf{FE.sk}_i := \mathsf{FE.sk}$ and we have

$$C_{\mathsf{s},2^i}(M, \widehat{\mathsf{K}}_i, \widehat{\mathsf{R}}_i) = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, U[\widehat{M_{2^i}}]; \widehat{\mathsf{R}}_i) = \mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]}.$$

From the above observation, we can see that B breaks the security of FE if A distinguishes the two games. This completes the proof of the lemma. □

Lemma 3.6.8. *We have* $|\Pr[\mathsf{E}_{i,3}] - \Pr[\mathsf{E}_{i,4}]| = \mathrm{negl}(\lambda)$.

Proof. Due to the change we introduced, $\widehat{\mathsf{K}}_i$ is not used to answer the encryption queries any more and used only when generating $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)$ in $\mathbf{Game}_{i,3}$. We also observe that $\widehat{\mathsf{R}}_i$ is used only when generating $\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]}$. Therefore, the lemma follows by straightforward reductions to the security of PRF. □

Lemma 3.6.9. *We have* $|\Pr[\mathsf{E}_{i,4}] - \Pr[\mathsf{E}_{i,5}]| = \mathrm{negl}(\lambda)$.

Proof. To prove the lemma, let us assume that $|\Pr[\mathsf{E}_{i,4}] - \Pr[\mathsf{E}_{i,5}]|$ is non-negligible and construct an adversary B that breaks the selective security of ABE using A. B proceeds as follows.

Setup phase. At the beginning of the game, B inputs 1^λ to A and obtains 1^{s} and $X \subset \Sigma^*$ from A. Then, B sets $X_i := \{\hat{\mathbf{x}} = \mathbf{x} \| \bot^{2^i - |\mathbf{x}|} : \mathbf{x} \in X,\ 2^{i-1} < |\mathbf{x}| \leq 2^i\}$ and submits its target X_i and $(1^\lambda, 1^{2^i \eta}, 1^{\hat{\mathsf{d}}})$ to its challenger. Then, the challenger samples

$$(\mathsf{ABE.mpk}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{2^i \eta}, 1^{\hat{\mathsf{d}}})$$

and returns $\mathsf{ABE.mpk}$ to B. B then sets $\mathsf{ABE.mpk}_i := \mathsf{ABE.mpk}$. In the rest of the simulation, it implicitly sets $\mathsf{ABE.msk}_i := \mathsf{ABE.msk}$ without knowing the value. It then chooses $\widehat{\mathsf{K}}_j, \mathsf{R}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for $j \in [0, \lambda] \backslash \{i\}$ and $(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{out}(\lambda)}, 1^{\mathsf{d}(\lambda)})$ for $j \in [0, \lambda]$. It also computes $\mathsf{FE.sk}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i, C_{\mathsf{s},2^i})$.

B then handles the the encryption and key queries as follows.

Encryption queries. Given two messages $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, $\mathcal{B}$ sets $\ell := |\mathbf{x}|$ and $i' = \lceil \log \ell \rceil$. If $i' \neq i$, B answers the encryption query using

$(\widehat{\mathsf{K}}_{i'}, \mathsf{R}_{i'}, \mathsf{FE.mpk}_{i'}, \mathsf{FE.msk}_{i'})$. Otherwise, $\mathcal{B}$ makes an encryption query for the attribute $\hat{\mathbf{x}} = \mathbf{x} \| \bot^{2^i - \ell}$ and messages $(m^{(0)}, m^{(1)})$ to its challenger. Then, the challenger runs

$$\mathsf{ABE.ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, \hat{\mathbf{x}}, \mathsf{m}^{(\mathsf{b})})$$

and returns a ciphertext $\mathsf{ABE.ct}$ to B. Then, it returns $\mathsf{NfaABE.ct} = (\mathsf{FE.sk_i}, \mathsf{ABE.mpk_i}, \mathsf{ABE.ct})$ to A. Here, B uses $\mathsf{FE.sk}_i$ that is sampled in the setup phase.

Key queries. Given an NFA M of size s from A, B first chooses $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for $j \in [0, \lambda] \backslash \{i\}$. It then queries a secret key for $U[\widehat{M}_{2^i}]$ to its challenger. Then, the challenger runs

$$\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, U[\widehat{M}_{2^i}])$$

and returns $\mathsf{ABE.sk}_{U[\widehat{M}_{2^i}]}$ to B. It then computes $\mathsf{FE.ct_j}$ for $j \in [0, \lambda]$ as Eq. (3.4) and returns $\mathsf{NfaABE.sk}_M := \{\mathsf{FE.ct_j}\}_{\mathsf{j} \in [0,\lambda]}$ to A.

Output phase: B outputs the same bit as A as its guess.

It is easy to see that B simulates $\mathbf{Game}_{i,4}$ if $b = 0$ and $\mathbf{Game}_{i,5}$ if $b = 1$. Therefore, B breaks the security of ABE if A distinguishes the two games. It remains to prove that B is a legitimate adversary (i.e., it does not make any prohibited key queries). For any attribute $\hat{\mathbf{x}}$ for which B makes an encryption query and for any circuit $U[\widehat{M}_{2^i}]$ for which B makes a key query, we have

$$U[\widehat{M}_{2^i}](\hat{\mathbf{x}}) = \widehat{M}_{2^i}(\hat{\mathbf{x}}) = M(\mathbf{x}),$$

where the second equality above follows from Item 1 of Theorem 3.6.1. Therefore, B is a legitimate adversary as long as so is A. This completes the proof of the lemma. □

Lemma 3.6.10. *We have* $|\Pr[\mathsf{E}_{i,5}] - \Pr[\mathsf{E}_{i,6}]| = \mathrm{negl}(\lambda)$.

Proof. This follows as in the indistinguishability of $\mathbf{Game}_{i,0}$ and $\mathbf{Game}_{i,4}$, but in the reverse order. That is, we first change the random bits used in $\mathsf{ABE.KeyGen}$ to a pseudorandom one by invoking the security of PRF. We then generate $\mathsf{FE.ct_i}$ by using $\mathsf{FE.Enc}$ instead of Sim by invoking the full-simulation security of FE. Finally, we change the random bits used in $\mathsf{ABE.KeyGen}$ to a pseudorandom one by invoking the security of PRF again. □

This concludes the proof of Theorem 3.6.4.

□

3.6.5 Extensions

In Appendix B.2, we adapt our ABE construction to achieve (restricted versions of) attribute privacy. In more detail, we construct secret key predicate encryption and bounded key functional encryption for nondeterministic finite automata. In Appendix B.4, we additionally achieve machine privacy, improving the result of [Agrawal and Singh (2017)]. Intuitively, these results proceed by replacing the "inner" circuit ABE scheme in our compiler by predicate encryption or bounded key functional encryption scheme and arguing that the requisite efficiency requirements (Theorem 3.5.9) are not violated. Please see appendices B.2,B.4 for details.

3.7 Attribute based Encryption for NFA with Unbounded Size Machines and Inputs

In this section we construct a secret-key attribute-based encryption scheme (SKABE) for nondeterministic finite automata of arbitrary sizes supporting inputs of arbitrary length. We denote our scheme by $\mathsf{uNfaABE} = (\mathsf{uNfaABE.Setup}, \mathsf{uNfaABE.KeyGen}, \mathsf{uNfaABE.Enc}, \mathsf{uNfaABE.Dec})$ and its construction uses the following two ingredients.

1. $\mathsf{NfaABE} = (\mathsf{NfaABE.Setup}, \mathsf{NfaABE.KeyGen}, \mathsf{NfaABE.Enc}, \mathsf{NfaABE.Dec})$: An SKABE for NFA supporting inputs of *unbounded length* but for *bounded size* machines. We instantiate NfaABE from our construction in Section 3.6.2.
2. $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$: An ABE scheme for circuits that satisfies the efficiency properties described in Theorem 3.5.9. We can instantiate ABE with the scheme proposed by [Boneh *et al.* (2014)].
3. $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: a pseudorandom function, where a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ defines a function $\mathsf{PRF.Eval}(\mathsf{K}, \cdot) : \{0,1\}^\lambda \to \mathcal{R}$, where we assume $\mathcal{R}$ to be the randomness space of *both* $\mathsf{NfaABE.Setup}$ and $\mathsf{ABE.Setup}$ algorithms. Note that without loss of generality, we may assume $\mathcal{R} = \{0,1\}^{p(\lambda)}$ for some sufficiently large polynomial $p(\lambda)$.

Below we provide our construction for SKABE for NFA.

3.7.1 Construction of $\mathsf{uNfaABE}$

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

1. Sample two PRF keys $\mathsf{K_{NfaABE}} \leftarrow \mathsf{PRF.Setup}(1^\lambda), \mathsf{K_{ABE}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
2. Output $\mathsf{uNfaABE.msk} = (\mathsf{K_{NfaABE}}, \mathsf{K_{ABE}})$.

$\mathsf{uNfaABE.Enc}(\mathsf{uNfaABE.msk}, \mathbf{x}, m)$: On input the master secret key $\mathsf{uNfaABE.msk}$, an attribute as $\mathbf{x} \in \Sigma^*$ of length at most 2^λ and a message $m \in \mathcal{M}$, do the following:

1. Parse the master secret key as $\mathsf{uNfaABE.msk} = (\mathsf{K_{NfaABE}}, \mathsf{K_{ABE}})$. Denote $\ell = |\mathbf{x}|$.
2. For all $i \in [\ell]$, do the following:
   (a) Sample $\mathsf{NfaABE.msk}_i \leftarrow \mathsf{NfaABE.Setup}(1^\lambda, 1^i; r_i)$ as an NfaABE master secret key, where $r_i = \mathsf{PRF.Eval}(\mathsf{K_{NfaABE}}, \mathsf{i})$.

   Note that i denotes the size of the NFAs that are supported by $\mathsf{NfaABE.msk}_i$.

   (b) Compute $\mathsf{NfaABE.ct}_i = \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_i}, \mathbf{x}, \mathsf{m}, 1^{\mathsf{i}})$.
3. Sample $(\mathsf{ABE.mpk}_\ell, \mathsf{ABE.msk}_\ell) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^\ell, 1^{\hat{\mathsf{d}}}; r_\ell)$ as an ABE key pair, where $r_\ell = \mathsf{PRF.Eval}(\mathsf{K_{ABE}}, \ell)$.

   Note that ℓ and $\hat{\mathsf{d}}$ denote the input length and the depth of the circuit respectively that $(\mathsf{ABE.mpk}_\ell, \mathsf{ABE.msk}_\ell)$ supports.
4. Compute $\mathsf{ABE.ct}_\ell = \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m})$.
5. Output $\mathsf{uNfaABE.ct} = (\{\mathsf{NfaABE.ct_i}\}_{\mathsf{i}\in[\ell]}, \mathsf{ABE.mpk}_\ell, \mathsf{ABE.ct}_\ell)$.

$\mathsf{uNfaABE.KeyGen}(\mathsf{uNfaABE.msk}, M)$: On input the master secret key $\mathsf{uNfaABE.msk}$ and the description of a NFA $M = (Q, \Sigma, T, q_{\mathsf{st}}, F)$, proceed as follows.

1. Parse the master secret key as $\mathsf{uNfaABE.msk} = (\mathsf{K_{NfaABE}}, \mathsf{K_{ABE}})$. Denote $\mathsf{s}{=}|M|$.
2. For all $i \in [\mathsf{s}]$, do the following:
   (a) Let $\widehat{M_i} = \mathsf{To\text{-}Circuit}_{\mathsf{s},i}(M)$. (See Theorem 3.6.1 for the definition of To-Circuit.)

   (b) Sample $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^i, 1^{\hat{\mathsf{d}}}; r_i)$ as an ABE key pair, where $r_i = \mathsf{PRF.Eval}(\mathsf{K_{ABE}}, \mathsf{i})$.

   (c) Compute $\mathsf{ABE.sk}_i = \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i, \widehat{M_i})$.

Note that $\forall i \in [\mathsf{s}]$, i and $\hat{\mathsf{d}}$ denotes the input length and the depth of the circuit respectively that $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)$ supports.

3. Sample $\mathsf{NfaABE.msk_s} \leftarrow \mathsf{NfaABE.Setup}(1^\lambda, 1^{\mathsf{s}}; r_{\mathsf{s}})$ as an NfaABE master secret key, where $r_{\mathsf{s}} = \mathsf{PRF.Eval}(\mathsf{K}_{\mathsf{NfaABE}}, \mathsf{s})$.

4. Compute $\mathsf{NfaABE.sk_s} = \mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk_s}, M)$.

5. Output $\mathsf{uNfaABE.sk}_M = (\mathsf{NfaABE.sk_s}, \{\mathsf{ABE.mpk}_i, \mathsf{ABE.sk}_i\}_{i \in [\mathsf{s}]})$.

$\mathsf{uNfaABE.Dec}(\mathsf{uNfaABE.sk}_M, M, \mathsf{uNfaABE.ct}, \mathbf{x})$: On input a secret key for NFA M and a ciphertext encoded under some attribute $\mathbf{x}$, proceed as follows:

1. Parse the secret key as $\mathsf{uNfaABE.sk}_M = (\mathsf{NfaABE.sk}_{|M|}, \{\mathsf{ABE.mpk}_i, \mathsf{ABE.sk}_i\}_{i \in [|M|]})$ and the ciphertext as $\mathsf{uNfaABE.ct} = (\{\mathsf{NfaABE.ct_i}\}_{\mathsf{i} \in [|\mathbf{x}|]}, \mathsf{ABE.mpk}_{|\mathbf{x}|}, \mathsf{ABE.ct}_{|\mathbf{x}|})$.

2. If $|\mathbf{x}| \geq |M|$, compute and output $\mathsf{NfaABE.Dec}(\mathsf{NfaABE.sk}_{|M|}, M, \mathsf{NfaABE.ct}_{|\mathsf{M}|}, \mathbf{x})$.

3. Otherwise, compute and output $\mathsf{ABE.Dec}(\mathsf{ABE.mpk}_{|\mathbf{x}|}, \mathsf{ABE.sk}_{|\mathbf{x}|}, \widehat{M}_{|\mathbf{x}|}, \mathsf{ABE.ct}_{|\mathbf{x}|}, \mathbf{x})$, where $\widehat{M}_{|\mathbf{x}|} = \mathsf{To\text{-}Circuit}_{|M|,|\mathbf{x}|}(M)$.

3.7.2 Correctness of uNfaABE

The following theorem asserts that our scheme is efficient.

Theorem 3.7.1. *The scheme* $\mathsf{uNfaABE} = (\mathsf{uNfaABE.Setup}, \mathsf{uNfaABE.KeyGen}, \mathsf{uNfaABE.Enc}, \mathsf{uNfaABE.Dec})$ *defined above runs in polynomial time, as long as* $\hat{\mathsf{d}}$ *and* $|\Sigma|$ *are polynomials in* λ. .

Proof. It is easy to see that the $\mathsf{uNfaABE.Setup}$ runs in polynomial time.

We start with showing that $\mathsf{uNfaABE.KeyGen}$ runs in polynomial time. Note that for any NFA M with size $\mathsf{s} = |M|$ and any input length $i < [\mathsf{s}]$, Item 2 of Theorem 3.6.1 asserts that the sizes and depths of $\mathsf{To\text{-}Circuit}_{\mathsf{s},i}$ and $\widehat{M}_i = \mathsf{To\text{-}Circuit}_{\mathsf{s},i}(M)$ are both bounded by $\mathrm{poly}(\mathsf{s}, i)$ and $\mathrm{poly}(\log \mathsf{s}, \log i)$ respectively. Hence, the efficiency requirements met by [Boneh *et al.* (2014)] discussed in Theorem 3.5.9 which is used to instantiate ABE implies that $\mathsf{ABE.Setup}$ and $\mathsf{ABE.KeyGen}$ always runs in time polynomial in λ, s and $\hat{\mathsf{d}}$. Further, the efficiency of NfaABE discussed in Theorem 3.6.2 which is used to instantiate NfaABE implies that $\mathsf{NfaABE.Setup}$ and $\mathsf{NfaABE.KeyGen}$ runs in polynomial time.

Next, we show that $\mathsf{uNfaABE.Enc}$ runs in polynomial time. We have a similar reasoning as above to argue the following: for any input $\mathbf{x}$ with length $\ell = |\mathbf{x}|$ and any NFA size $i \in [\ell]$, the efficiency of NfaABE discussed in Theorem 3.6.2 implies that $\mathsf{NfaABE.Setup}$ and $\mathsf{NfaABE.Enc}$ run in polynomial time.

We finally bound the running time of $\mathsf{uNfaABE.Dec}$. It is easy to see that the efficiency guarantees given by the underlying schemes NfaABE and ABE along with Theorem 3.6.1 implies that $\mathsf{uNfaABE.Dec}$ runs in polynomial time. $\square$

The following theorem addresses the correctness of the scheme.

Theorem 3.7.2. *For appropriately chosen* $\hat{\mathsf{d}} = \hat{\mathsf{d}}(\lambda)$*, our scheme* $\mathsf{uNfaABE}$ *is correct for any NFA.*

Proof. As long as $\hat{\mathsf{d}}$ is chosen appropriately, we can argue the correctness of $\mathsf{uNfaABE}$ as follows. The $\mathsf{uNfaABE.Dec}$ algorithm takes as input

$$\begin{aligned}\mathsf{uNfaABE.sk}_M &= \big(\mathsf{NfaABE.sk}_{|M|}, \{\mathsf{ABE.mpk}_i, \mathsf{ABE.sk}_i\}_{i\in[|M|]}\big) \text{ and}\\ \mathsf{uNfaABE.ct} &= (\{\mathsf{NfaABE.ct}_i\}_{i\in[|\mathbf{x}|]}, \mathsf{ABE.mpk}_{|\mathbf{x}|}, \mathsf{ABE.ct}_{|\mathbf{x}|})\end{aligned}$$

as a secret key for an NFA M and a ciphertext under an attribute $\mathbf{x}$ respectively.

By our definition, for any $i \in \mathbb{N}, \mathsf{NfaABE.msk}_i$ supports NFA machines of size bounded above by i with unbounded input length. Thus, in the case when $|\mathbf{x}| \geq |M|$, the correctness of NfaABE scheme implies the correctness of the $\mathsf{uNfaABE}$ scheme.

Alternatively, we have by Theorem 3.5.9 that for any $i \in \mathbb{N}, (\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)$ supports a circuit class of input length i and depth $\hat{\mathsf{d}} = \hat{\mathsf{d}}(\lambda)$ while Theorem 3.6.1 implies that $\mathsf{depth}(\widehat{M_i})$ is always bounded above by $\mathrm{poly}(\log i, \log \mathsf{s}) = \mathrm{poly}(\log \lambda) < \lambda$. In particular, we have that $\mathsf{depth}(\widehat{M_i}) < \hat{\mathsf{d}}$, where we set $\hat{\mathsf{d}} = \lambda$ and thus the circuit $\widehat{M_i}$ is supported by the ABE scheme. Therefore, in the case when $|\mathbf{x}| < |M|$, the correctness of ABE scheme implies the correctness of the $\mathsf{uNfaABE}$ scheme. $\square$

3.7.3 Proof of Security for $\mathsf{uNfaABE}$

Next, we prove that the above $\mathsf{uNfaABE}$ scheme is secure, as long as the underlying NfaABE and ABE schemes are secure.

Theorem 3.7.3. *Assume that* NfaABE *and* ABE *both satisfy selective indistinguishability based security and* PRF *is a secure pseudorandom function. Then,* $\mathsf{uNfaABE}$ *satisfies selective security.*

Proof. To show that any PPT adversary A succeeds with only negligible probability in the selective security game of $\mathsf{uNfaABE}$ as stated in Definition 3.5.4, let us introduce the following sequence of games $\{\mathbf{Game}_k\}_{k\in[0,6]}$ between the $\mathsf{uNfaABE}$ challenger and A.

$\mathbf{Game}_0$**:** For any challenge message query $(m^{(0)}, m^{(1)})$ with respect to any attribute $\mathbf{x}$, this game corresponds to the real experiment where the $\mathsf{uNfaABE}$ challenger encrypts messages corresponding to challenge bit $b = 0$.

$\mathbf{Game}_1$**:** In this game, we change the setup phase as follows.

Setup phase. Given $X \subset \Sigma^*$ from A, the challenger chooses $\mathsf{uNfaABE.msk} \leftarrow \mathsf{NfaABE.Setup}(1^\lambda)$ as in the previous game. In addition, it precomputes all relevant master keys as follows which are later used to answer encryption as well as key queries.

$$\begin{aligned} \mathsf{NfaABE.msk}_i &\leftarrow \mathsf{NfaABE.Setup}(1^\lambda, 1^i; r_i), \text{ where } r_i = \mathsf{PRF.Eval}(\mathsf{K}_{\mathsf{NfaABE}}, \mathsf{i}) \\ (\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) &\leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^i, 1^{\hat{\mathsf{d}}}; r'_i), \text{ where } r'_i = \mathsf{PRF.Eval}(\mathsf{K}_{\mathsf{ABE}}, \mathsf{i}) \end{aligned}$$

for all $i \in [T], T = \max(\ell_{\max}, \mathsf{s}_{\max})$, where we define $\ell_{\max} := \max\{|\mathbf{x}| : \mathbf{x} \in X\}$ as per Definition 3.5.4 and $\mathsf{s}_{\max}$ as the maximum size of any NFA queried by A. Note that both $\ell_{\max}$ and $\mathsf{s}_{\max}$ and hence T are bounded by the running time of A and thus are polynomial in λ.

$\mathbf{Game}_2$**:** This game is exactly the same as $\mathbf{Game}_1$ except that now the challenger samples the master keys in setup phase for *both* the underlying schemes NfaABE and ABE using true randomness instead of using PRF randomness as

$$\begin{aligned} \{\mathsf{NfaABE.msk}_i &\leftarrow \mathsf{NfaABE.Setup}(1^\lambda, 1^i)\}_{i\in[T]}, \\ \{(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) &\leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^i, 1^{\hat{\mathsf{d}}})\}_{i\in[T]} \end{aligned}$$

$\mathbf{Game}_3$**:** In this game, for any challenge message query $(m^{(0)}, m^{(1)})$ with respect to any attribute $\mathbf{x}$, the challenger generates NfaABE ciphertexts corresponding to challenge bit $b = 1$ while the ABE ciphertexts are generated corresponding to challenge bit $b = 0$.

$\mathbf{Game}_4$**:** In this game, for any challenge message query $(m^{(0)}, m^{(1)})$ with respect to any attribute $\mathbf{x}$, the challenger generates both NfaABE and ABE ciphertexts corresponding to challenge bit $b = 1$.

$\mathbf{Game}_5$**:** This game is exactly the same as $\mathbf{Game}_4$ except that now the challenger samples the master keys in the setup phase for *both* the underlying schemes NfaABE and ABE using PRF randomness again instead of using true randomness.

$\mathbf{Game}_6$**:** For any challenge message query $(m^{(0)}, m^{(1)})$ with respect to any attribute $\mathbf{x}$, this game corresponds to the real experiment where the $\mathsf{uNfaABE}$ challenger encrypts messages corresponding to challenge bit $b = 1$.

In the following, let E_k denote the event that A outputs 1 in $\mathbf{Game}_k$. To prove the theorem, we will show that $|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_6]| = \mathrm{negl}(\lambda)$, since $\mathbf{Game}_0$ (resp., $\mathbf{Game}_6$) corresponds to the selective security game with $b = 0$ (resp., $b = 1$). Since we have

$$|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_6]| \leq \sum_{k\in[0,5]} |\Pr[\mathsf{E}_k] - \Pr[\mathsf{E}_{k+1}]|$$

by the triangle inequality, it suffices to show $|\Pr[\mathsf{E}_k] - \Pr[\mathsf{E}_{k+1}]| = \mathrm{negl}(\lambda)$ for all $k \in [0, 5]$. In order to prove indistinguishability between $\mathbf{Game}_2$ and $\mathbf{Game}_3$ (resp., $\mathbf{Game}_3$ and $\mathbf{Game}_4$), we further introduce a sequence of games $\{\mathbf{Game}_{2,i}\}_{i\in[0,\ell_{\max}]}$ (resp., $\{\mathbf{Game}_{3,i}\}_{i\in[0,\ell_{\max}]}$), where $\ell_{\max}$ was defined in $\mathbf{Game}_1$. We then describe the two games $\mathbf{Game}_{2,i}$ and $\mathbf{Game}_{3,i}$ for any $i \in [0, \ell_{\max}]$ as follows.

$\mathbf{Game}_{2,i}$**:** The game proceeds as follows.

Setup phase. At the beginning of the game, A takes 1^λ as input and submits the set of its target $X \subset \Sigma^*$ to the $\mathsf{uNfaABE}$ challenger. The challenger then generates the master keys as before.

The challenger answers the encryption and key queries made by A as follows.

Encryption queries. For any message pair $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, the challenger sets $\ell = |\mathbf{x}|$, chooses the appropriate NfaABE and ABE keys from the set of precomputed keys and computes

$$\mathsf{NfaABE.ct_j} \leftarrow \begin{cases} \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_j}, \mathbf{x}, \mathsf{m}^{(0)}, 1^{\mathsf{j}}) & \text{If } j > i \\ \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_j}, \mathbf{x}, \mathsf{m}^{(1)}, 1^{\mathsf{j}}) & \text{If } j \leq i \end{cases}$$

for $j \in [\ell]$. It also computes $\mathsf{ABE.ct}_\ell \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(0)})$ and returns $\mathsf{uNfaABE.ct} \leftarrow (\{\mathsf{NfaABE.ct_j}\}_{\mathsf{j}\in[\ell]}, \mathsf{ABE.mpk}_\ell, \mathsf{ABE.ct}_\ell)$ to A.

Key queries. Given an NFA M from A, the challenger sets $\mathsf{s} = |M|$ and then uses the precomputed master keys to compute and returns $\mathsf{uNfaABE.sk}_M = (\mathsf{NfaABE.sk_s}, \{\mathsf{ABE.mpk}_j, \mathsf{ABE.sk}_j\}_{j\in[\mathsf{s}]})$ to A.

Finally, A outputs its guess b'.

$\mathsf{Game}_{3,i}$: The game proceeds as follows.

Setup phase. At the beginning of the game, A takes 1^λ as input and submits the set of its target $X \subset \Sigma^*$ to the $\mathsf{uNfaABE}$ challenger. The challenger then generates the master keys as before.

The challenger answers the encryption and key queries made by A as follows.

Encryption queries. For any message pair $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, the challenger sets $\ell = |\mathbf{x}|$, chooses the appropriate NfaABE and ABE from the set of precomputed keys and computes

$$\mathsf{ABE.ct}_\ell \leftarrow \begin{cases} \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(0)}) & \text{If } \ell > i \\ \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(1)}) & \text{If } \ell \leq i. \end{cases}$$

It also computes $\{\mathsf{NfaABE.ct_j} \leftarrow \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_j}, \mathbf{x}, \mathsf{m}^{(1)}, 1^{\mathsf{j}})\}_{\mathsf{j}\in[\ell]}$ and returns $\mathsf{uNfaABE.ct} = (\{\mathsf{NfaABE.ct_j}\}_{\mathsf{j}\in[\ell]}, \mathsf{ABE.mpk}_\ell, \mathsf{ABE.ct}_\ell)$ to A.

Key queries. Given an NFA M from A, the challenger sets $\mathsf{s} = |M|$ and then uses the precomputed master keys to compute and returns $\mathsf{uNfaABE.sk}_M = (\mathsf{NfaABE.sk_s}, \{\mathsf{ABE.mpk}_j, \mathsf{ABE.sk}_j\}_{j\in[\mathsf{s}]})$ to A.

Finally, A outputs its guess b'.

We first note that $\forall k \in [2,3], \mathbf{Game}_k = \mathbf{Game}_{k,0}$ and $\mathbf{Game}_{k,\ell_{\max}} = \mathbf{Game}_{k+1}$. Since we have

$$|\Pr[\mathsf{E}_k] - \Pr[\mathsf{E}_{k+1}]| = |\Pr[\mathsf{E}_{k,0}] - \Pr[\mathsf{E}_{k,\ell_{\max}}]| \le \sum_{i=0}^{(\ell_{\max}-1)} |\Pr[\mathsf{E}_{k,i}] - \Pr[\mathsf{E}_{k,i+1}]|$$

by the triangle inequality, it suffices to show $|\Pr[\mathsf{E}_{k,i}] - \Pr[\mathsf{E}_{k,i+1}]| = \mathrm{negl}(\lambda)$, for all $k \in [2,3]$ and for all $i \in [0, \ell_{\max} - 1]$. To complete the proof of the theorem, it remains to prove the following lemmas.

Lemma 3.7.4. *We have* $\Pr[\mathsf{E}_0] = \Pr[\mathsf{E}_1]$.

Proof. The change introduced here is only conceptual, where all the master keys $\{\mathsf{NfaABE.msk}_i\}_{i\in[T]}$ and $\{(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i)\}_{i\in[T]}$ are computed a-priori. The lemma trivially follows. $\square$

Lemma 3.7.5. *We have* $|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| = \mathrm{negl}(\lambda)$.

Proof. Note that the PRF keys $\mathsf{K}_{\mathsf{NfaABE}}$ and $\mathsf{K}_{\mathsf{ABE}}$ are only used for generating randomness in order to precompute the NfaABE and ABE master keys and are not used anywhere else in $\mathbf{Game}_1$. Further, note that since the two PRF keys $\mathsf{K}_{\mathsf{NfaABE}}$ and $\mathsf{K}_{\mathsf{ABE}}$ are *independent* of each other we can replace them at once with true randomness in $\mathbf{Game}_2$. Therefore, the lemma follows by a straightforward reduction to the security of PRF with respect to the two independently drawn keys $\mathsf{K}_{\mathsf{NfaABE}}$ and $\mathsf{K}_{\mathsf{ABE}}$. $\square$

Lemma 3.7.6. *We have* $|\Pr[\mathsf{E}_{2,i}] - \Pr[\mathsf{E}_{2,i+1}]| = \mathrm{negl}(\lambda), \forall i \in [0, \ell_{\max} - 1]$.

Proof. To prove the lemma, let us assume that $|\Pr[\mathsf{E}_{2,i}] - \Pr[\mathsf{E}_{2,i+1}]|$ is non-negligible and construct an adversary B that breaks the selective security of NfaABE using A. B proceeds as follows.

Setup phase. At the beginning of the game, B inputs 1^λ to A and obtains $X \subset \Sigma^*$ from A. Then, B submits $(1^\lambda, 1^i)$ and X_i to the NfaABE challenger where $X_i := \{\mathbf{x} \mid \mathbf{x} \in X \text{ and } |\mathbf{x}| = i\}$ and precomputes $\{(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j)\}_{j\in[T]}$ and $\{\mathsf{NfaABE.msk}_j\}_{j\in[T]\setminus\{i\}}$. It then implicitly sets $\mathsf{NfaABE.msk}_i := \mathsf{NfaABE.msk}$ without knowing its value, where $\mathsf{NfaABE.msk}$ is chosen by the NfaABE challenger and further handles the encryption and key queries as follows.

Encryption queries. Given a message pair $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, B sets $\ell = |\mathbf{x}|$, chooses $(\mathsf{ABE.mpk}_\ell, \mathsf{ABE.msk}_\ell)$ and $\{\mathsf{NfaABE.msk}_j\}_{j\in[\ell]\setminus\{i\}}$ from the set of precomputed ABE and NfaABE keys to answer any encryption query as follows.

1. B computes $\{\mathsf{NfaABE.ct_j} \leftarrow \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_j}, \mathbf{x}, \mathsf{m}^{(0)}, 1^{\mathsf{j}})\}_{\mathsf{j}\in[\mathsf{i}+1,\ell]}$.
2. B makes an encryption query $\big((m^{(0)}, m^{(1)}), \mathbf{x}\big)$ to the NfaABE challenger. The challenger computes
$$\mathsf{NfaABE.ct} \leftarrow \begin{cases} \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \mathbf{x}, \mathsf{m}^{(0)}, 1^{\mathsf{i}}) & \text{If } b = 0 \\ \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk}, \mathbf{x}, \mathsf{m}^{(1)}, 1^{\mathsf{i}}) & \text{If } b = 1 \end{cases}$$
and returns $\mathsf{NfaABE.ct}$ to B. B then sets $\mathsf{NfaABE.ct_i} := \mathsf{NfaABE.ct}$.
3. B computes $\{\mathsf{NfaABE.ct_j} \leftarrow \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_j}, \mathbf{x}, \mathsf{m}^{(1)}, 1^{\mathsf{j}})\}_{\mathsf{j}\in[1,\mathsf{i}-1]}$.

B also computes by itself $\mathsf{ABE.ct}_\ell \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(0)})$ and returns $\mathsf{uNfaABE.ct} = (\{\mathsf{NfaABE.ct_j}\}_{\mathsf{j}\in[\ell]}, \mathsf{ABE.mpk}_\ell, \mathsf{ABE.ct}_\ell)$ to A.

Key queries. Given an NFA M from A, B first sets $\mathsf{s} = |M|$, computes $\{\widehat{M_j} = \mathsf{To\text{-}Circuit}_{\mathsf{s},j}(M)\}_{j\in[\mathsf{s}]}$ and then chooses $\{(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j)\}_{j\in[\mathsf{s}]}$ from the set of precomputed ABE keys. It then proceeds as follows.

1. If $\mathsf{s} \neq i$, B chooses $\mathsf{NfaABE.msk_s}$ from the set of precomputed NfaABE keys to compute $\mathsf{NfaABE.sk_s} \leftarrow \mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk_s}, M)$.
2. If $\mathsf{s} = i$, B submits M to the NfaABE challenger upon which the challenger computes and returns $\mathsf{NfaABE.sk} \leftarrow \mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk}, M)$. B then sets $\mathsf{NfaABE.sk_s} := \mathsf{NfaABE.sk}$.

B also computes by itself $\{\mathsf{ABE.sk}_j \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j, \widehat{M_j})\}_{j\in[\mathsf{s}]}$ and returns $\mathsf{uNfaABE.sk}_M = \big(\mathsf{NfaABE.sk_s}, \{\mathsf{ABE.mpk}_j, \mathsf{ABE.sk}_j\}_{j\in[\mathsf{s}]}\big)$ to A.

Output phase: B outputs the same bit as A as its guess.

It is easy to see that B simulates $\mathbf{Game}_{2,i}$ if $b = 0$ and $\mathbf{Game}_{2,i+1}$ if $b = 1$. Therefore, B breaks the security of NfaABE if A distinguishes the two games. It remains to prove that B is a legitimate adversary (i.e., it does not make any prohibited key queries). For any attribute $\mathbf{x}$ for which B makes an encryption query and for any key query for an NFA M, we have that $M(\mathbf{x}) = 0$, by the legitimacy of adversary A.

From the above observation, we can see that B breaks the security of NfaABE if A distinguishes the two games. This completes the proof of the lemma. □

Lemma 3.7.7. *We have* $|\Pr[\mathsf{E}_{3,i}] - \Pr[\mathsf{E}_{3,i+1}]| = \text{negl}(\lambda), \forall i \in [0, \ell_{\max} - 1]$.

Proof. To prove the lemma, let us assume that $|\Pr[\mathsf{E}_{3,i}] - \Pr[\mathsf{E}_{3,i+1}]|$ is non-negligible and construct an adversary B that breaks the selective security of ABE using A. B proceeds as follows.

Setup phase. At the beginning of the game, B inputs 1^λ to A and obtains $X \subset \Sigma^*$. Then, B submits $(1^\lambda, 1^i, 1^{\hat{\mathsf{d}}})$ and X_i to the ABE challenger, where $X_i := \{\mathbf{x} \mid \mathbf{x} \in X \text{ and } |\mathbf{x}| = i\}$. The ABE challenger chooses $(\mathsf{ABE.mpk}, \mathsf{ABE.msk})$ and replies to B with $\mathsf{ABE.mpk}$ upon which B precomputes $\{(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j)\}_{j \in [T] \setminus \{i\}}$ and $\{\mathsf{NfaABE.msk}_j\}_{j \in [T]}$ and then implicitly sets $(\mathsf{ABE.mpk}_i, \mathsf{ABE.msk}_i) := (\mathsf{ABE.mpk}, \mathsf{ABE.msk})$ without knowing the value of $\mathsf{ABE.msk}$. It then handles the encryption and key queries as follows.

Encryption queries. Given a message pair $(m^{(0)}, m^{(1)})$ and $\mathbf{x} \in X$ from A, B sets $\ell = |\mathbf{x}|$, chooses $\{\mathsf{NfaABE.msk}_j\}_{j \in [\ell]}$ and $\mathsf{ABE.mpk}_\ell$ from the set of precomputed NfaABE and ABE keys to answer any encryption query as follows.

1. If $\ell > i$, B computes $\mathsf{ABE.ct}_\ell \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(0)})$.
2. If $\ell = i$, B makes a challenge ciphertext query $\big((m^{(0)}, m^{(1)}), \mathbf{x}\big)$ to the ABE challenger. The challenger computes

$$\mathsf{ABE.ct} \leftarrow \begin{cases} \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(0)}) & \text{If } b = 0 \\ \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(1)}) & \text{If } b = 1 \end{cases}$$

   and returns $\mathsf{ABE.ct}$ to B. B then sets $\mathsf{ABE.ct}_\ell := \mathsf{ABE.ct}$.
3. If $\ell < i$, B computes $\mathsf{ABE.ct}_\ell \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}_\ell, \mathbf{x}, \mathsf{m}^{(1)})$.

B also computes $\{\mathsf{NfaABE.ct_j} \leftarrow \mathsf{NfaABE.Enc}(\mathsf{NfaABE.msk_j}, \mathbf{x}, \mathsf{m}^{(1)}, 1^{\mathsf{j}})\}_{\mathsf{j} \in [\ell]}$ and returns to A $\mathsf{uNfaABE.ct} = (\{\mathsf{NfaABE.ct_j}\}_{\mathsf{j} \in [\ell]}, \mathsf{ABE.mpk}_\ell, \mathsf{ABE.ct}_\ell)$.

Key queries. Given an NFA M from A, B first sets $\mathsf{s} = |M|$, computes $\{\widehat{M_j} = \mathsf{To\text{-}Circuit}_{\mathsf{s},j}(M)\}_{j \in [\mathsf{s}]}$ and then chooses $\mathsf{NfaABE.msk_s}$ from the set of precomputed NfaABE keys. It then proceeds as follows.

1. B chooses $\{(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j)\}_{j \in [\mathsf{s}] \setminus \{i\}}$ from the set of precomputed ABE keys to compute $\{\mathsf{ABE.sk}_j \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}_j, \mathsf{ABE.msk}_j, \widehat{M_j})\}_{j \in [\mathsf{s}] \setminus \{i\}}$.
2. B submits $\widehat{M_i}$ to the ABE challenger upon which the challenger computes and returns $\mathsf{ABE.sk} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mpk}, \mathsf{ABE.msk}, \widehat{M_i})$. B then sets $\mathsf{ABE.sk}_i := \mathsf{ABE.sk}$.

B also computes by itself $\mathsf{NfaABE.sk_s} \leftarrow \mathsf{NfaABE.KeyGen}(\mathsf{NfaABE.msk_s}, M)$ and returns to A $\mathsf{uNfaABE.sk}_M = \big(\mathsf{NfaABE.sk_s}, \{\mathsf{ABE.mpk}_j, \mathsf{ABE.sk}_j\}_{j\in[\mathsf{s}]}\big)$.

Output phase: B outputs the same bit as A as its guess.

It is easy to see that B simulates $\mathbf{Game}_{3,i}$ if $b = 0$ and $\mathbf{Game}_{3,i+1}$ if $b = 1$. Therefore, B breaks the security of ABE if A distinguishes the two games. It remains to prove that B is a legitimate adversary (i.e., it does not make any prohibited key queries). Note that any key query issued by B is a circuit $\widehat{M_i}$ against a key query for an NFA M issued by A. Further, Theorem 3.6.1 implies that for any NFA M, $M(\mathbf{x}) = \widehat{M_i}(\mathbf{x}), \forall \mathbf{x} \in \Sigma^{\leq i}$. Thus, for any attribute $\mathbf{x}$ of length i for which B makes an encryption query and for any NFA key query M issued by A, we have $\widehat{M_i}(\mathbf{x}) = M(\mathbf{x}) = 0$, by the legitimacy of A.

From the above observation, we can see that B breaks the security of ABE if A distinguishes the two games. This completes the proof of the lemma. □

Lemma 3.7.8. *We have* $|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_5]| = \mathrm{negl}(\lambda)$.

Proof. The proof follows similarly to Lemma 3.7.5. □

Lemma 3.7.9. *We have* $\Pr[\mathsf{E}_5] = \Pr[\mathsf{E}_6]$.

Proof. The proof follows similarly to Lemma 3.7.4. □

This concludes the proof of Theorem 3.7.3. □

3.8 FE for DFA implies iO

Here, we show that secret key functional encryption (SKFE) for DFA with security against unbounded collusion implies indistinguishability obfuscation (iO). This result illuminates the difficulty of constructing such SKFE from a standard assumption, since no construction of iO from standard assumption is known despite the significant research effort in recent years [Gorbunov *et al.* (2013); Garg *et al.* (2013*b*,*a*); Gorbunov *et al.* (2015, 2012); Agrawal and Rosen (2017); Goldwasser *et al.* (2013*a*); Gorbunov *et al.* (2015); Agrawal (2017); Abdalla *et al.* (2015); Agrawal *et al.* (2016); Lin (2017); Baltico *et al.* (2017); Ananth and Jain (2015); Bitansky and Vaikuntanathan (2015); Ananth *et al.*

(2015*b*); Lin (2016); Lin and Vaikuntanathan (2016); Lin (2017); Agrawal and Singh (2017); Lin and Tessaro (2017); Ananth *et al.* (2018); Lin and Matt, (2018); Agrawal (2019)].

3.8.1 Preliminaries on DFA and Branching Programs

Here, we first recall that a deterministic finite automaton (DFA) is a special case of NFA where for the transition function T, $T(\sigma, q)$ consists of a single element in Q for any $\sigma \in \Sigma$ and $q \in Q$. We then define branching program similarly to [Brakerski and Vaikuntanathan (2014)].

Definition 3.8.1 (Branching Programs)**.** A width-5 permutation branching program BP of length L with input space $\{0,1\}^\ell$ is a sequence of L tuples of the form $(\mathsf{var}(t), \sigma_{t,0}, \sigma_{t,1})$ where

- $\mathsf{var} : [L] \to [\ell]$ is a function that associates the t-th tuple with an input bit $x_{\mathsf{var}(t)}$.
- $\sigma_{j,0}$ and $\sigma_{j,1}$ are permutations on 5 elements. We will think of $\sigma_{j,0}$ and $\sigma_{j,1}$ as bijective functions from the set $\{1,2,3,4,5\}$ to itself.

The computation of the program BP on input $\mathbf{x} = (x_1, \ldots, x_\ell)$ proceeds as follows. The state of the computation at any point in time t is a number $\zeta_t \in \{1,2,3,4,5\}$. Computation starts with the initial state $\zeta_0 = 1$. The state ζ_t is computed recursively as

$$\zeta_t = \sigma_{t, x_{\mathsf{var}(t)}} (\zeta_{t-1}) \,. \tag{3.6}$$

Finally, after L steps, our state is ζ_L. The output of the computation $\mathsf{BP}(\mathbf{x})$ is 1 if $\zeta_L = 1$ and 0 otherwise.

We will use the following theorem, which essentially says that an NC^1 circuit can be converted into an equivalent branching program.

Theorem 3.8.2 (Barrington's Theorem [Barrington (1989)])**.** *Every Boolean NAND circuit C that acts on ℓ inputs and has depth d can be computed by a width-5 permutation branching program BP of length 4^d. Given the description of the circuit BP, the description of the branching program BP can be computed in $\mathrm{poly}(\ell, 4^d)$ time. In particular, if C is a polynomial-sized circuit with logarithmic depth (i.e., if the circuit is in NC^1), BP can be computed in polynomial time.*

3.8.2 SKFE for DFA implies iO

We first state and prove the following theorem.

Theorem 3.8.3. *Let $d = d(\lambda)$ and $\ell = \ell(\lambda)$ be integers. There exist deterministic algorithms* Encode *and* ToDFA *with the following properties.*

- $\mathsf{Encode}(\mathbf{x}) \to \mathbf{y} \in \{0,1\}^n$*, where $\mathbf{x} \in \{0,1\}^\ell$ and n is a parameter determined by d and ℓ.*
- $\mathsf{ToDFA}(C) \to M$*, where $C : \{0,1\}^\ell \to \{0,1\}$ is a circuit with depth bounded by d and M is a DFA over alphabet $\Sigma = \{0,1\}$.*

We have that $M(\mathbf{y}) = 1$ if and only if $C(\mathbf{x}) = 1$. We also have that the running time of Encode *and* ToDFA *is* $\mathrm{poly}(\ell, 2^d)$*. In particular, if C is a polynomial-sized circuit with logarithmic depth (i.e., if the circuit is in* NC^1*),* Encode *and* $\mathsf{ToDFA}(C)$ *run in polynomial time.*

Proof. We define Encode and ToDFA as follows:

- $\mathsf{Encode}(\mathbf{x})$ outputs $\mathbf{y} := (\mathbf{x}\|0)^L$, where $(\mathbf{x}\|0)^L$ denotes the L-times repetition of the same string $\mathbf{x}\|0$ and $L = 4^d$.
- $\mathsf{ToDFA}(C)$ first converts C into a branching program $\mathsf{BP} = \{(\mathsf{var}(t), \sigma_{t,0}, \sigma_{t,1})\}_{t\in[L]}$ of length $L = 4^d$ by using the algorithm in Theorem 3.8.2. It then outputs DFA $M = (Q, \Sigma, T, q_{\mathsf{st}}, F)$, where

  $$Q = \{q_{i,j}^{(k)}\}_{i\in[L],j\in[0,\ell],k\in[5]} \cup \{q_{\mathsf{ed}}^{(k)}\}_{k\in[5]}, \quad \Sigma = \{0,1\}, \quad q_{\mathsf{st}} = q_{1,0}^{(1)}, \quad F = \{q_{\mathsf{ed}}^{(1)}\},$$

  and

  $$T(b, q_{i,j}^{(k)}) = \begin{cases} q_{i,\mathsf{var}(i)}^{\sigma_{i,b}(k)} & \text{if } j = \mathsf{var}(i) - 1 \\ q_{i+1,0}^{(k)} & \text{if } j = \ell \wedge i \neq L \\ q_{\mathsf{ed}}^{(k)} & \text{if } j = \ell \wedge i = L \\ q_{i,j+1}^{(k)} & \text{if } j \neq \mathsf{var}(i) - 1,\ j \neq \ell, \end{cases} \qquad T(b, q_{\mathsf{ed}}^{(k)}) = q_{\mathsf{ed}}^{(k)}.$$

In the following, we often denote $q_{L+1,0}^{(k)} := q_{\mathsf{ed}}^{(k)}$ for notational convenience.

It is easy to see that the running time of the algorithms is bounded by $\mathrm{poly}(\ell, 2^d)$. We then prove $M(\mathbf{y}) = C(\mathbf{x})$. Since we have $C(\mathbf{x}) = \mathsf{BP}(\mathbf{x})$ by Theorem 3.8.2, it suffices to show $\mathsf{BP}(\mathbf{x}) = M(\mathbf{y})$. To show this, we prove the following claim.

Claim 3.8.4. *Let t be an integer $t \in [0, L]$. Then, state q_{st} goes to state $q_{t+1,0}^{\zeta_t}$ on input $(\mathbf{x}\|0)^t$, where ζ_t is defined as Eq.* (3.6).

Proof. The proof is by induction on t. For the base case of $t = 0$, the statement holds because $q_{\mathsf{st}} = q^{(1)}_{1,0}$ and $\zeta_0 = 1$. Next, we prove the statement for $t = t^*$ assuming that the statement for $t = t^* - 1$. By the assumption, state q_{st} goes to state $q^{\zeta_{t^*-1}}_{t^*,0}$ on input $(\mathbf{x}\|0)^{t^*-1}$. It suffices to prove that state $q^{\zeta_{t^*-1}}_{t^*,0}$ goes to state $q^{\zeta_{t^*}}_{t^*+1,0}$ on input $\mathbf{x}\|0$. By inspection, it can be seen that state $q^{\zeta_{t^*-1}}_{t^*,0}$ goes to state $q^{\zeta_{t^*-1}}_{t^*,\mathsf{var}(t^*)-1}$ on input $x_1, \ldots, x_{\mathsf{var}(t^*)-1}$. We also observe that state $q^{\zeta_{t^*-1}}_{t^*,\mathsf{var}(t^*)-1}$ goes to state

$$q^{\sigma_{t^*,\mathsf{var}(t^*)}(\zeta_{t^*-1})}_{t^*,\mathsf{var}(t^*)} = q^{\zeta_{t^*}}_{t^*,\mathsf{var}(t^*)}$$

on $x_{\mathsf{var}(t^*)}$, where the above equality follows from the definition of ζ_{t^*}. Again by inspection, it can be seen that state $q^{\zeta_{t^*}}_{t^*,\mathsf{var}(t^*)}$ goes to state $q^{\zeta_{t^*}}_{t^*,\ell}$ on $x_{\mathsf{var}(t^*)+1}, \ldots, x_\ell$ and $q^{\zeta_{t^*}}_{t^*,\ell}$ goes state $q^{\zeta_{t^*}}_{t^*+1,0}$ on input 0. This completes the proof of the claim. $\square$

By setting $t = L$, the claim implies that state q_{st} goes to state $q^{\zeta_L}_{\mathsf{ed}}$ on input $\mathbf{y}$. Thus, we have $M(\mathbf{y}) = 1$ iff $\zeta_1 = 1$, which implies $M(\mathbf{y}) = 1$ iff $\mathsf{BP}(\mathbf{x}) = 1$. This completes the proof of the theorem. $\square$

We then discuss that if there exists subexponentially secure SKFE (please see Appendix B.1.3 for a formal definition) for DFA that is very selectively secure against unbounded collusion, it can be converted into a secure indistinguishability obfuscation.

To do so, we first convert an SKFE for DFA into an SKFE for NC^1 circuits. The latter SKFE has the same setup algorithm as the former, but when generating a secret key for a circuit C, it first converts C into a DFA M using the algorithm in Theorem 3.8.3 and then invokes the key generation algorithm of the SKFE for DFA on input M. Similarly, when encrypting a message $\mathbf{x}$, it computes $\mathbf{y}$ as in Theorem 3.8.3 and then invokes the encryption algorithm of the SKFE for DFA on input $\mathbf{y}$. The decryption algorithm is defined naturally. It is easy to see that this conversion preserves the correctness and the security since we have $M(\mathbf{y}) = C(\mathbf{x})$ by Theorem 3.8.3.

Then, we apply the conversion given by [Ananth and Jain (2015); Bitansky and Vaikuntanathan (2015)] to the SKFE for NC^1 to obtain SKFE for all circuits. We then further apply the conversion by Kitagawa et al. [Kitagawa *et al.* (2017, 2018*a*)] to the SKFE for all circuits to obtain iO. Note that while the former conversion incurs only polynomial loss, the latter conversion incurs sub-exponential security loss.

In summary, we obtain the following theorem.

Theorem 3.8.5. *If there exists a subexponentially secure SKFE scheme for DFA that is* *very selectively secure against unbounded collusion, then there exists an indistinguishability obfuscation.*

CHAPTER 4

Attribute based Encryption for Deterministic Finite Automata from Standard Static Assumptions

4.1 Introduction

In this chapter we continue our study of attribute based encryption (ABE) but shift the attention towards supporting deterministic finite automata (DFA) from static assumptions over bilinear maps.

As mentioned in Chapter 3, [Waters (2012)] provided the first construction of ABE for regular languages. Here, the secret key is associated with a DFA and ciphertext is associated with attribute $\mathbf{x}$ of *arbitrary* length. The same secret key can directly decrypt ciphertexts that encode inputs of varying lengths, yielding the first ABE that supports a *uniform* model of computation. While the construction in [Waters (2012)] relied on hardness assumptions over bilinear maps, which are well understood, the assumption is *parametrized* (also known as "q-type"), which means that the size of the assumption depends on the queries made by the adversary. Achieving a construction of ABE for DFA from standard static assumptions over bilinear maps had remained elusive since then. Although our results in Chapter 3 provides a *secret key* ABE for *nondeterministic* finite automata, our construction makes use of highly lattice specific machinery (such as reusable garbled circuits [Goldwasser *et al.* (2013*a*)]) relying on the learning with errors assumption. Therefore, it is unclear how to use these ideas to improve the state of affairs in the world of pairings.

4.2 Our Contributions

In this work, we construct the first *public key* ABE scheme for DFA from static assumptions on pairings, namely, the DLIN assumption. Our scheme supports unbounded length inputs as well as unbounded length machines. In more detail, secret keys in our construction are associated with a DFA M of unbounded length, ciphertexts are associated with a tuple $(\mathbf{x}, m)$ where $\mathbf{x}$ is a public attribute of unbounded length and m is a secret message bit, and decryption recovers m if and only if $M(\mathbf{x}) = 1$. Our construction also supports unbounded key requests by the adversary. Additionally, via a simple tweak to our construction, we also obtain the first ciphertext-policy ABE for DFA from the DLIN assumption.

We contrast our results with prior work in Table 4.1. For brevity, we only compare with constructions of ABE that support uniform models of computation (in particular, handle unbounded input lengths) and rely on standard assumptions. Other relevant and concurrent work is discussed in Sections 4.4 and 4.5.

Construction	**Model**	KP **or** CP	**Number of Keys**	**Assumption**	**Security**
[Waters (2012)]	DFA	KP	unbounded	q-type assumption on bilinear maps	Selective
[Attrapadung (2014)]	DFA	KP & CP	unbounded	q-type assumption on bilinear maps	Adaptive
[Agrawal and Singh (2017)]	DFA	KP	single	LWE	Selective
[Agrawal *et al.* (2019*a*)]	NFA	KP	unbounded	LWE	Selective
[Gong *et al.* (2019)]	DFA	KP	unbounded	kLIN	Selective
This	DFA	KP & CP	unbounded	DLIN	Selective*

Table 4.1: Comparison with prior work supporting unbounded input length. KP and CP indicate key-policy and ciphertext-policy respectively.

4.3 Our Techniques.

A natural starting point for constructing (key policy) ABE for DFA is (key policy) ABE for monotone span programs (MSP), which has been studied extensively in the literature. Recall that an MSP is specified by a pair $(\mathbf{L}, \rho)$ of a matrix and a labelling function where $\mathbf{L} \in \mathbb{Z}_p^{\ell \times m}$, $\rho : [\ell] \to \{0, 1\}^*$ for some integer ℓ, m. Intuitively, the map ρ labels row i with attribute $\rho(i)$. Given a set of attributes I as input, the MSP accepts the input iff the sub-matrix of $\mathbf{L}$ restricted to attributes selected by I contains a special target vector in its row span (please see Section 4.7.1 for the precise definition).

Step 1: Leveraging ABE for MSP. Our first observation is that DFA computation is simple enough to be encoded into an MSP. In more detail, given a DFA machine M and an input string $\mathbf{x}$, it is possible to map the DFA M into an MSP $(\mathbf{L}_M, \rho_M)$ and the input $\mathbf{x}$ into a set of attributes $S_\mathbf{x}$ such that the MSP $(\mathbf{L}_M, \rho_M)$ accepts attributes $S_\mathbf{x}$ iff $M(\mathbf{x}) = 1$. We exhibit such a map in Section 4.9.1 and prove the following theorem:

Theorem 4.3.1. *(Informal) Let $(\mathbf{L}_M, \rho_M)$ be the MSP and $S_\mathbf{x}$ be the set of attributes obtained by applying the map specified in Section 4.9.1 to M and $\mathbf{x}$ respectively. Then, the MSP $(\mathbf{L}_M, \rho_M)$ accepts attributes $S_\mathbf{x}$ iff $M(\mathbf{x}) = 1$.*

This provides a starting point for using ABE for MSP, which can be constructed from static assumptions, as a building block towards constructing ABE for DFA.

Step 2: Handling Unbounded Length. While this seems promising as a first step, the careful reader may have noticed that the above idea fails to address the primary challenge of supporting DFA, namely, that of handling inputs of unbounded length. DFA is a uniform model of computation, which means that the same machine must process inputs of arbitrary length. On the other hand, an MSP can only process inputs of bounded length – in particular, the length of inputs that an MSP can read is clearly bounded above by the number of rows in $\mathbf{L}$.

This appears to make ABE for MSP almost useless for our purposes, since there is no way to guarantee that $|\mathbf{x}|$ is less than the number of rows in $\mathbf{L}$ (denoted by $|\mathbf{x}| \leq |M|$ in the sequel[1]). However, notice that since both the inputs and the machines have unbounded length, it still holds in some cases that $|\mathbf{x}| \leq |M|$, and if we can handle this, it still constitutes progress. More hurdles present themselves – for instance, the syntax of ABE for DFA does not allow the setup algorithm to know the lengths $|\mathbf{x}|$, $|M|$, the key generation algorithm cannot know $|\mathbf{x}|$ and the encrypt algorithm cannot know $|M|$.

[1]While imprecise, we use this notation here for intuition. Formally, it will turn out to be sufficient to compare $|\mathbf{x}|$ with $|Q|$, where $|Q|$ is the number of states in M.

But this challenge can be overcome by making use of the so called *unbounded* ABE schemes, as described next.

Unbounded ABE schemes (for MSP) [Okamoto and Takashima (2012); Chen *et al.* (2018)] are those in which the setup algorithm places no restriction on the length of the attributes or the size of the policies that are embedded in the ciphertexts and keys. Moreover, the key generation and encrypt algorithms do not require knowledge of input length or policy size respectively. While significantly more challenging to build than their bounded counterparts, a small number of existing constructions [Okamoto and Takashima (2012); Chen *et al.* (2018)] achieve this property while relying on standard assumptions.

We show in Section 4.8.2 that unbounded key policy ABE schemes for MSP can indeed be used to construct ABE for DFA so long as $|\mathbf{x}| \leq |M|$. More formally, we define relation $R^{\mathsf{KP}}(S, (\mathbf{L}, \rho)) = 1$ iff the span program $(\mathbf{L}, \rho)$ accepts the attribute set S and $R^{\mathsf{DFA}\leq}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{\leq} |M|\big)$. Then, we have that:

Theorem 4.3.2. *(Informal) Let* kpABE *be a secure unbounded ABE for the relation* R^{KP}. *Then, the construction* $\mathsf{DfaABE}^{\leq}$ *provided in Section 4.8.2 is a secure ABE for the relation* $R^{\mathsf{DFA}\leq}$.

Step 3: The trick in Chapter 3. To construct a full fledged ABE for DFA, our next tool is a technique used in Chapter 3. We showed how to construct an ABE for nondeterministic finite automata (NFA) that supports unbounded inputs and unbounded machines, by running in parallel two restricted ABE for NFA schemes: one that supports unbounded inputs but bounded machines and one that supports bounded inputs but unbounded machines.

Our goal is to construct an ABE scheme DfaABE for the relation $R^{\mathsf{DFA}}(\mathbf{x}, M) = M(\mathbf{x})$. By using a similar technique, we can construct our DfaABE from two special ABE schemes as follows:

1. An ABE $\mathsf{DfaABE}^{\leq}$ for the relation $R^{\mathsf{DFA}\leq}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{\leq} |M|\big)$.
2. An ABE $\mathsf{DfaABE}^{>}$ for the relation $R^{\mathsf{DFA}>}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{>} |M|\big)$.

It is easy to see that given constructions for the special ABE schemes $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$, we may construct DfaABE simply by running them in parallel. In more detail, the setup algorithm of DfaABE simply runs the setup algorithms of the underlying special ABEs and outputs the public and master secret keys by combining their outputs, the encrypt algorithm encrypts its input $(\mathbf{x}, \mu)$ under both special ABEs, the key generation algorithm produces a key under both special ABEs and the decryption algorithm invokes the decryption of one or the other depending on whether $|\mathbf{x}| \overset{?}{\leq} |M|$. This intuition is formalized in Section 4.8.1, where we prove the following theorem:

Theorem 4.3.3. *(Informal) Assume that* $\mathsf{DfaABE}^{\leq}$ *and* $\mathsf{DfaABE}^{>}$ *are secure ABE schemes for relations* $R^{\mathsf{DFA}\leq}$ *and* $R^{\mathsf{DFA}>}$ *respectively. Then, the scheme* DfaABE *constructed in Section 4.8.1 is a secure ABE for relation* R^{DFA}.

Step 4: Plugging the gap with ciphertext policy ABE. We already constructed an ABE for the case of $|\mathbf{x}| \leq |M|$. The case of $|\mathbf{x}| > |M|$ is more challenging, since to use ABE for MSP, it is necessary that the MSP be large enough to read the input as we have discussed above. To handle this, we simply switch the role of key generator and encryptor! In more detail, if the encryptor could instead embed $\mathbf{x}$ into an MSP and the key generator could embed M into a set of attributes, then the dilemma of compatible sizes could be resolved and we would be back in business. We show that this can be done; we provide a map in Section 4.9.2 that achieves this embedding. More formally, we prove that:

Theorem 4.3.4. *Let* $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}})$ *be the MSP and* S_M *be the set of attributes obtained by applying the map specified in Section 4.9.2 to* $\mathbf{x}$ *and* M *respectively. Then, the MSP* $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}})$ *accepts attributes* S_M *iff* $M(\mathbf{x}) = 1$.

In order to support encryption of an MSP $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}})$, we now need an unbounded *ciphertext policy* ABE for MSP. In more detail, we define $R^{\mathsf{CP}}((\mathbf{L}, \rho), S) = 1$ iff the span program $(\mathbf{L}, \rho)$ accepts the attribute set S. Recall that $R^{\mathsf{DFA}>}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{>} |M|\big)$. Then, we show in Section 4.8.3 that:

Theorem 4.3.5. *(Informal.) Let* cpABE *be a secure unbounded ABE scheme for the relation* R^{CP}*. Then the construction* $\mathsf{DfaABE}^{>}$ *provided in Section 4.8.3 is a secure ABE for the relation* $R^{\mathsf{DFA}>}$.

To summarize, our approach is based on the observation that we must only construct an MSP of length $\max(|\mathbf{x}|, |M|)$, where $|\mathbf{x}|$ is known to the encryptor and $|M|$ is known to the key generator (and neither know the other). When the input vector has size $|\mathbf{x}| \leq |M|$, we embed the DFA into a monotone span program which has number of rows proportional to $|M|$, and the input into a set of attributes – this ensures that the MSP is large enough to support an input of length $|\mathbf{x}|$. We may then leverage an unbounded kpABE scheme to handle this case. On the other hand, when $|\mathbf{x}| > |M|$, we instead embed the input vector into a monotone span program which has number of rows proportional to $|\mathbf{x}|$, and the machine into a set of attributes – this again ensures that the MSP is large enough to support an input of size $|M|$. We may then leverage an unbounded cpABE scheme to handle this case. Of course, neither party knows which case it must support, so it simply provides information for both and leaves it to the decryptor to make the choice!

Step 5: Instantiating the kpABE and cpABE. Finally, we must ensure that we can instantiate unbounded ABE schemes kpABE and cpABE for the relations R^{KP} and R^{CP} that we require. While prior work provides constructions of unbounded key policy and ciphertext policy ABE schemes for MSP, these unfortunately cannot be plugged into our compiler out of the box. This is because our construction requires the ABE schemes to support "multi-use" of attributes, i.e. when the map ρ in the MSP is not restricted to be injective. Moreover, the ABE schemes are required to be unbounded, as already discussed above. Finally, we want the schemes to be proven secure from static assumptions such as DLIN, not from q-type assumptions. Schemes achieving all these properties do not exist in the literature to the best of our knowledge.[2] Hence, we must refashion existing schemes to satisfy this. In Section C.1, we provide constructions for multi-use unbounded key policy and ciphertext policy ABE schemes by modifying the constructions in [Chen *et al.* (2018)]. Let R^{MUKP} and R^{MUCP} be the same relations as R^{KP} and R^{CP} defined above, but with the requirement that the underlying MSPs in both relations support multi-use of attributes. Then, we obtain the following theorem:

Theorem 4.3.6. *(Informal.) The constructions* kpABE *provided in Appendix C.1.2 and* cpABE *provided in Appendix C.1.4 are unbounded ABE schemes for the relations* R^{MUKP} *and* R^{MUCP} *respectively. Security of* kpABE *relies on the* MDDH *assumption and security of* cpABE *relies on the* DLIN *assumption.*

[2] Only exception is the very recent construction by Kowalczyk and Wee [Kowalczyk and Wee, (2019)]. However, their scheme can only deal with NC_1 circuit instead of general MSP and thus our embedding of DFA into MSP cannot be used.

For both KP and CP-ABE schemes, we simply modify the schemes in [Chen *et al.* (2018)] so that we allow multi-use of the same attribute in an MSP. However, this simple modification ruins the original security proof given by [Chen *et al.* (2018)] in both cases. The reason is that the core statistical argument in the security proof does not work any more in the multi-use setting. Intuitively, the problem is that the terms used as "one-time pads" in the single-use setting are used multiple times in the multi-use setting. In both KP and CP cases, we switch to weaker security notions than adaptive security and give security proofs by taking advantage of weaker setting.

For KP-ABE scheme, we prove semi-adaptive security. To prove the security, we first use the handy bilinear entropy expansion lemma [Chen *et al.* (2018)] to create an instance of a multi-use variant of the KP-ABE scheme by [Lewko *et al.* (2010)] (hereafter denoted by LOSTW) in the semi-functional space. To give a proof, we decompose the LOSTW secret key into smaller pieces and gradually add semi-functional randomness to them through a hybrid argument in a way that their distribution depends on the challenge attribute, in a similar manner to [Agrawal and Chase (2016)]. Since this step requires the knowledge of the challenge attribute, we can only prove semi-adaptive security of the scheme. Intuitively, because of this decomposition, we use the "one-time pad" only single time in one hybrid game and can avoid getting into the aforementioned problem of using one-time pads multiple times. Finally, we can use the core statistical step similarly to the case of single-use setting.

For CP-ABE scheme, we prove the security notion that we call selective* security, where the adversary is forced to choose its key queries and the challenge attribute after seeing the master public key. The first step of the proof is similar to the KP-ABE case. Namely, we first use the bilinear entropy expansion lemma [Chen *et al.* (2018)] to create an instance of the LOSTW CP-ABE scheme in the semi-functional space. However, in the next step, we cannot use the above decomposition idea due to technical reasons, which in turn prohibits us from using the statistical argument in the core step. We overcome this by using computational argument instead, which uses the DLIN assumption instead. The idea of using computational argument here was taken from some of prior works [Lewko and Waters (2012); Attrapadung (2014, 2016)].

Putting together these pieces yields our final result – a key-policy ABE for DFA that supports unbounded inputs, unbounded machines and unbounded key requests.

Ciphertext Policy ABE for DFA. In the above description, note that our construction DfaABE uses the underlying kpABE and cpABE in a symmetric way. Thus, by swapping the use of kpABE and cpABE in our construction, we can equivalently construct ciphertext policy ABE for DFA.

In more detail, we exchange the maps used by KeyGen and Enc in the constructions of $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$ in Sections 4.8.2 and 4.8.3. Please see Section 4.10 for more details. Thus, we obtain

Theorem 4.3.7. *There exists a secure key-policy and ciphertext-policy ABE for R^{DFA} from the* DLIN *assumption.*

Efficiency. Our ABE schemes inherit their efficiency from the underlying building blocks of embedding DFA computations into MSPs. In particular, the ciphertexts and secret keys in our KP-ABE for DFA scale as $O(|\mathbf{x}|^3)$ and $O(|Q|^2)$ respectively. Accordingly, the CP-ABE for DFA being obtained in a symmetric way, its ciphertexts scale as $O(|Q|^2)$ and $O(|\mathbf{x}|^3)$.

4.4 Related Work.

In this section, we discuss the related work in the area, categorized by hardness assumptions. We begin with constructions based on bilinear maps. The first construction of ABE for DFA was given by [Waters (2012)] as discussed above. This scheme achieved selective security, which was improved to adaptive by Attrapadung (2014). For span programs, there have been many constructions [Lewko and Waters (2010); Okamoto and Takashima (2010); Lewko *et al.* (2010); Lewko and Waters (2011); Lewko (2012); Okamoto and Takashima (2012); Rouselakis and Waters (2013); Chen and Wee (2013, 2014); Wee (2015); Attrapadung (2014); Chen *et al.* (2015*a*); Attrapadung *et al.* (2015); Kowalczyk and Lewko (2015); Attrapadung (2016); Agrawal and Chase (2017); Chen *et al.* (2018)] that achieve various tradeoffs between security (selective versus adaptive), assumptions (static versus parametrized), underlying mathematical structure (prime versus composite order groups), policy embedding (key versus ciphertext policy) and efficiency. In this work, we are particularly concerned with unbounded ABE schemes, in particular those by [Okamoto and Takashima (2012); Chen *et al.* (2018)].

From the Learning With Errors assumption (LWE), [Boyen and Li (2015)] provided a construction of ABE for DFA, but this was restricted to DFAs with *bounded* length inputs, rendering moot the primary advantage of a DFA over circuits. Recently, [Ananth *et al.* (2019)] provided an ABE for random access machines from LWE, but this construction is also restricted to inputs of bounded length. [Agrawal and Singh (2017)] constructed a primitive closely related to ABE for DFA, namely *reusable garbled DFA* from LWE, but their construction is only secure in the single key setting, namely, where the adversary is limited to requesting a single function key. In contrast, we support unbounded key requests in this work.

From strong assumptions such as the the existence of multilinear maps [Garg *et al.* (2013*a*)], witness encryption [Goldwasser *et al.* (2013*a*)] or indistinguishability obfuscation [Barak *et al.* (2001); Garg *et al.* (2013*a*)], attribute based encryption (or its more powerful generalization – *functional encryption*) has been constructed even for Turing machines [Ananth and Sahai (2017); Agrawal and Maitra (2018); Kitagawa *et al.* (2019)], but these are not considered standard assumptions; indeed many candidate constructions have been broken [Cheon *et al.* (2015); Coron *et al.* (2015); Hu and Jia, (2015); Cheon *et al.* (2016*a*,*b*); Miles *et al.* (2016); Coron *et al.* (2017); Apon *et al.* (2017)].

Also relevant to our work are the constructions of [Brakerski and Vaikuntanathan (2016); Goyal *et al.* (2016)], which provide attribute based encryption for the so called "bundling functionalities". Here, the size of the public parameters does not depend on the length of the input (say ℓ) chosen by the encryptor. However, the key generator must generate a key for a circuit with a fixed input length (say ℓ'), and decryption only succeeds if $\ell = \ell'$. Thus, bundling functionalities do not capture the essential challenge of supporting dynamic data sizes as discussed in [Goyal *et al.* (2016)].

4.5 Concurrent Work.

We note that a concurrent work by [Gong *et al.* (2019)] constructs KP-ABE scheme for DFA relying on the k-LIN assumption. Although there is a qualitative overlap in our final results as shown in Table 4.1, the approaches and techniques in their work are quite different from ours. They construct KP-ABE from scratch imitating the transition function of a DFA using bilinear maps directly. This, in turn, yields a scheme with better concrete efficiency and security than ours. In particular, in the KP-ABE setting, our ciphertexts and keys scale as $O(|\mathbf{x}|^3)$ and $O(|Q|^2)$ respectively while the ciphertexts and keys in Gong *et al.* (2019) scale linearly as $O(|\mathbf{x}|)$ and $O(|Q|)$ respectively. Also, our construction achieves selective* security based on DLIN assumption, while their construction achieves selective security and relies on the slightly weaker k-LIN assumption. On the other hand, our scheme is a generic compiler, and has conceptual advantages: our construction is modular and simpler and yields CP-ABE essentially for free. Further, it reduces the question of adaptive security for DFA for both KP-ABE and CP-ABE to that of adaptive security for unbounded KP-ABE and CP-ABE for MSP from static assumptions.

4.6 Organization

We organize the rest of the chapter as follows. In Section 4.7, we provide the definitions and preliminaries we require. In Section 4.8, we provide our ABE for DFA supporting unbounded input and unbounded machines from kpABE and cpABE for monotone span programs. In Section 4.9, we describe how to encode DFA computation into a monotone span program (MSP): Section 4.9.1 shows the encoding procedure for any DFA machine to a MSP (and DFA input to attribute set) while Section 4.9.2 shows the encoding procedure for any input string to a MSP (and DFA machine to attribute set). In Appendix C, we instantiate our ingredient kpABE and cpABE using techniques from [Chen *et al.* (2018)]. In Section 4.10 we put together all ingredients to instantiate our ABE for DFA.

4.7 Preliminaries

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

4.7.1 Definitions: Monotone Span Programs

A monotone span program (MSP) over $\mathbb{Z}_p$ is specified by a pair $(\mathbf{L}, \rho)$ of a matrix and a labelling function where

$$\mathbf{L} \in \mathbb{Z}_p^{\ell \times m} \qquad\qquad \rho : [\ell] \to \mathbb{Z}$$

for some integer ℓ, m. Intuitively, the map ρ labels row i with attribute $\rho(i)$.

A span program takes as input a set of integers and accepts or rejects an input by the following criterion. Let $S = \{u_1, \ldots, u_t\} \subseteq \mathbb{Z}$ be a set of integers. Intuitively, each u_i represents some attribute. For the set S, we define another set $I \subseteq [\ell]$ as $I = \{i \in [\ell] : \rho(i) \in S\}$ and $\mathbf{L}_I$ as the submatrix of $\mathbf{L}$ restricted to set of rows I, i.e.

obtained by removing row j of $\mathbf{L}$ for any $j \notin I$. We say that

$$(\mathbf{L}, \rho) \text{ accepts } S \text{ iff } (1, 0, \ldots, 0) \text{ is in the row span of } \mathbf{L}_I.$$

We can write this also as $\mathbf{e}_1 \in \operatorname{span}(\mathbf{L}_I^\top)$.

4.7.2 Definitions: Deterministic Finite Automata

A Deterministic Finite Automaton (DFA) M is represented by the tuple $(Q, \Sigma, T, q_{\mathsf{st}}, F)$ 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. We say that M accepts $\mathbf{x} = (x_1, \ldots, x_k) \in \Sigma^k$ if there exists a sequence of states $q_1, \ldots, q_{k+1}$ such that $q_1 = q_{\mathsf{st}}$, $q_{i+1} \in T(x_i, q_i)$ for $i \in [k]$ and $q_{k+1} \in F$. We assume w.l.o.g. that the states are numbered as 1 to $|Q|$, i.e., $Q = \{1, 2, \ldots, |Q|\}$ with $q_{\mathsf{st}} = 1$ along with $\Sigma = \{0, 1\}$ and $F = \{|Q|\}$. Note that any DFA with many accepting states can be converted to a DFA with a single accepting state [3], and states may be renumbered so that the last state is the accepting one.

4.7.3 Definitions: Attribute-Based Encryption

Syntax. Let $R : A \times B \to \{0, 1\}$ be a relation where A and B denote “ciphertext attribute" and “key attribute” spaces. An attribute based encryption scheme for R is defined by the following PPT algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$**:** The setup algorithm takes as input the unary representation of the security parameter λ and outputs a master public key mpk and a master secret key msk.

$\mathsf{Encrypt}(\mathsf{mpk}, \mu, X) \to \mathsf{ct}$**:** The encryption algorithm takes as input a master public key mpk, the message bit μ, and a ciphertext attribute $X \in A$. It outputs a ciphertext ct.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{mpk}, Y) \to \mathsf{sk}_Y$**:** The key generation algorithm takes as input the master

[3]In more detail, we may map any input $\mathbf{x} \in \{0, 1\}^*$ to $\mathbf{x}\|\star$, where $\star$ is a special symbol, and modify M so that we change the accepting state to be $\{|Q| + 1\}$ and add edges from the previous accepting state to $|Q| + 1$, where edges are labelled with $\star$.

secret key msk, the master public key mpk, and a key attribute $Y \in B$. It outputs a private key sk_Y.

$\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{ct}, \mathsf{X}, \mathsf{sk_Y}, \mathsf{Y}) \to \mu$ **or** $\perp$**:** We assume that the decryption algorithm is deterministic. The decryption algorithm takes as input the master public key mpk, a ciphertext ct, ciphertext attribute $X \in A$, a private key sk_Y, and private key attribute Y. It outputs the message μ or $\perp$ which represents that the ciphertext is not in a valid form.

We require the standard correctness of decryption: for all λ, $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $X \in A, Y \in B$ such that $R(X, Y) = 1$, and $\mathsf{sk}_Y \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{mpk}, Y)$, we have $\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{Encrypt}(\mathsf{mpk}, \mu, X), X, \mathsf{sk}_Y, Y) = \mu$.

Security. We now define the security for an ABE scheme Π by the following game between a challenger and an attacker $\mathcal{A}$.

- At first, the challenger runs the setup algorithm and gives mpk to $\mathcal{A}$.
- Then $\mathcal{A}$ may adaptively make key-extraction queries. We denote this phase PHASE1. In this phase, if $\mathcal{A}$ submits $Y \in B$ to the challenger, the challenger returns $\mathsf{sk}_Y \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{mpk}, Y)$.
- At some point, $\mathcal{A}$ outputs two equal length messages μ_0 and μ_1 and challenge ciphertext attribute $X^\star \in A$. $X^\star$ cannot satisfy $R(X^\star, Y) = 1$ for any attribute Y such that $\mathcal{A}$ already queried private key for Y.
- Then the challenger flips a random coin $\beta \in \{0, 1\}$, runs $\mathsf{Encrypt}(\mathsf{mpk}, \mu_\beta, X^\star) \to \mathsf{ct}^\star$ and gives challenge ciphertext $\mathsf{ct}^\star$ to $\mathcal{A}$.
- In PHASE2, $\mathcal{A}$ may adaptively make queries as in PHASE1 with following added restriction: $\mathcal{A}$ cannot make a key-extraction query for Y such that $R(X^\star, Y) = 1$.
- At last, $\mathcal{A}$ outputs a guess β' for β.

We say that $\mathcal{A}$ succeeds if $\beta' = \beta$ and denote the probability of this event by $\Pr^{\mathsf{ABE}}_{\mathcal{A},\Pi}$. The advantage of an attacker $\mathcal{A}$ is defined as $\mathsf{A}^{\mathsf{ABE}}_{\mathcal{A},\Pi} = |\Pr^{\mathsf{ABE}}_{\mathcal{A},\Pi} - \frac{1}{2}|$. We say that Π is adaptively secure if $\mathsf{A}^{\mathsf{ABE}}_{\mathcal{A},\Pi}$ is negligible for all probabilistic polynomial time (PPT) adversary $\mathcal{A}$.

Weaker Security Notions. A weaker notion called selective security can be defined as in the above game with the exception that the adversary $\mathcal{A}$ has to choose the challenge ciphertext attribute $X^\star$ before the setup phase but private key queries $Y_1, \ldots, Y_k$ and choice of (μ_0, μ_1) can still be adaptive. The stronger notion of semi-adaptive security lets the adversary output the challenge ciphertext attribute $X^\star$ after seeing the public key but before making any key requests. The still weaker notion of very selective security requires the adversary to output the challenge ciphertext attribute and private key queries at the very start of the game. An intermediate notion to semi-adaptive and very selective, which we term selective*, allows the adversary to receive the public parameters in the first step, but it must specify the challenge ciphertext attribute and private key queries after this step.

ABE for DFA. We then define ABE for DFA by specifying the relation. We define $A^{\mathsf{DFA}} = \{0,1\}^*$ and B^{DFA} as the set of all DFA, also represented as strings over $\{0,1\}^*$. Furthermore, we define the relation $R^{\mathsf{DFA}} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ as $R^{\mathsf{DFA}}(\mathbf{x}, M) = M(\mathbf{x})$.

An ABE scheme for the relation R^{DFA} is said to be ABE for DFA. We further define $R^{\mathsf{DFA}\leq} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ as

$$R^{\mathsf{DFA}\leq}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{\leq} |Q|\big),$$

where $|Q|$ is the number of states in M. We also define $R^{\mathsf{DFA}>}$ analogously.

Unbounded ABE for MSP. Here, we define unbounded ABE for MSP. There are distinctions between “single-use" and “multi-use" as well as “key-policy" and “ciphertext-policy". We first define multi-use key-policy unbounded ABE by specifying the relation R^{MUKP}. To do so, we set $A^{\mathsf{MUKP}} := 2^{\mathbb{Z}}$ (i.e., the set of all subsets of $\mathbb{Z}$) and B^{MUKP} as the set of monotone span programs on $\mathbb{Z}_p$ for some prime p, and $R^{\mathsf{MUKP}}(S, (\mathbf{L}, \rho)) = 1$ iff the span program $(\mathbf{L}, \rho)$ accepts the set $S \in A^{\mathsf{MUKP}}$. An ABE for R^{MUKP} is said to be “multi-use key-policy unbounded ABE".

We also define single-use key-policy unbounded ABE by specifying the relation R^{SUKP}. We set $A^{\mathsf{SUKP}} := 2^{\mathbb{Z}}$ and B^{SUKP} as the set of monotone span programs $(\mathbf{L}, \rho)$ such that ρ is injective. We define $R^{\mathsf{SUKP}}(S, (\mathbf{L}, \rho)) = 1$ iff the span program $(\mathbf{L}, \rho)$ accepts the set S. Finally, we can define the ciphertext variant of the above ABE by specifying R^{SUCP} and R^{MUCP}, where we set $A^{\mathrm{xx}\mathsf{CP}} = B^{\mathrm{xx}\mathsf{KP}}$ and $B^{\mathrm{xx}\mathsf{CP}} = A^{\mathrm{xx}\mathsf{KP}}$ for $\mathrm{xx} \in \{\mathsf{SU}, \mathsf{MU}\}$ and define the relation analogously.

Unbounded ABE for MSP with polynomial-valued attributes. We can consider a restricted variant of unbounded ABE for MSP where the value of attributes being used is polynomially bounded. Here, we focus on the case of multi-use and key-policy. Other cases will be defined similarly. We define $A^{\mathsf{MUKP}'}$ and $B^{\mathsf{MUKP}'}$ as

$$A^{\mathsf{MUKP}'} = \left\{ (S, 1^{s_{\max}}) : S \subseteq \mathbb{Z}, s_{\max} = \max_{s \in S} |s| \right\} \qquad \text{and}$$

$$B^{\mathsf{MUKP}'} = \left\{ ((\mathbf{L}, \rho), 1^{\rho_{\max}}) : (\mathbf{L}, \rho) \text{ is a span program over } \mathbb{Z}_p,\ \rho_{\max} = \max_{i \in [\ell]} |\rho(i)| \right\}$$

We define $R^{\mathsf{MUKP}'}(S, (\mathbf{L}, \rho)) := R^{\mathsf{MUKP}}(S, (\mathbf{L}, \rho))$. Here, the reason why we append $1^{s_{\max}}$ to S is somewhat technical. This is to enforce the adversary in the security definition who declares $S \in A^{\mathsf{MUKP}'}$ as its target to choose attributes with polynomially bounded values. Because of the similar reason, we append $1^{\rho_{\max}}$ to $(\mathbf{L}, \rho)$.

For ease of readability in the remainder of the paper, we will overload notation and denote $A^{\mathsf{MUKP}'}$ and $B^{\mathsf{MUKP}'}$ as A^{MUKP} and B^{MUKP} respectively. However, all our constructions will satisfy the constraint of attribute values being polynomially bounded.

4.7.4 Embedding Lemma for ABE

Here, we introduce a useful lemma that describes a sufficient criterion for implication from an ABE for a given predicate to an ABE for another predicate. The lemma is introduced in [Boneh and Hamburg (2008)] and later formally proven in [Attrapadung *et al.* (2015)]. The presentation here follows that of [Attrapadung *et al.* (2015)] with some simplifications. The lemma is applicable to any relation family. We consider two relation families:

$$R^{\mathsf{F}} : A \times B \to \{0, 1\}, \qquad\qquad R^{\mathsf{F}'} : A' \times B' \to \{0, 1\}.$$

Suppose that there exist two efficient mappings $f_{\mathsf{e}} : A' \to A$ and $f_{\mathsf{k}} : B' \to B$ which map parameters, ciphertext attributes, and key attributes, respectively, such that for all $X' \in A', Y' \in B'$,

$$R^{\mathsf{F}'}(X', Y') = 1 \Leftrightarrow R^{\mathsf{F}}(f_{\mathsf{e}}(X'), f_{\mathsf{k}}(Y')) = 1. \tag{4.1}$$

We can then construct an ABE scheme $\Pi' = \{\mathsf{Setup}', \mathsf{Encrypt}', \mathsf{KeyGen}', \mathsf{Decrypt}'\}$ for predicate $R^{\mathsf{F}'}$ from an ABE scheme $\Pi = \{\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt}\}$ for predicate R^{F} as follows. Let $\mathsf{Setup}' = \mathsf{Setup}$ and

$$\mathsf{Encrypt}'(\mathsf{mpk}, \mu, X') = \mathsf{Encrypt}(\mathsf{mpk}, \mu, f_{\mathsf{e}}(X')),$$
$$\mathsf{KeyGen}'(\mathsf{msk}, \mathsf{mpk}, Y') = \mathsf{KeyGen}(\mathsf{msk}, \mathsf{mpk}, f_{\mathsf{k}}(Y')),$$

and $\mathsf{Decrypt}'(\mathsf{mpk}, \mathsf{ct}, \mathsf{X}', \mathsf{sk}_{\mathsf{Y}'}, \mathsf{Y}') = \mathsf{Decrypt}(\mathsf{mpk}, \mathsf{ct}, \mathsf{f_e}(\mathsf{X}'), \mathsf{sk}_{\mathsf{Y}'}, \mathsf{f_k}(\mathsf{Y}'))$.

Lemma 4.7.1 (Embedding lemma [Boneh and Hamburg (2008); Attrapadung *et al.* (2015)]). *If Π is correct and secure, then so is Π'. This holds for very selective, selective, selective* and adaptive security.*

Intuitively, the forward and backward directions of Relation (4.1) ensure that the correctness and the security are preserving, respectively.

4.8 Attribute-based Encryption for DFA

We construct an ABE scheme for DFA denoted by $\mathsf{DfaABE} = (\mathsf{DfaABE.Setup}, \mathsf{DfaABE.KeyGen}, \mathsf{DfaABE.Enc}, \mathsf{DfaABE.Dec})$. Following the notation of Section 4.7, we achieve this by constructing an ABE scheme for the relation $R^{\mathsf{DFA}} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ which is defined as $R^{\mathsf{DFA}}(\mathbf{x}, M) = M(\mathbf{x})$. Recall that A^{DFA} is the set of all input strings and B^{DFA} is the set of all DFA. Let $|Q|$ be the number of states in M. As described in Section 4.3, our construction relies on two special ABE for DFA as follows:

1. An ABE denoted by $\mathsf{DfaABE}^{\leq}$ for the relation $R^{\mathsf{DFA}\leq} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ defined as:
$$R^{\mathsf{DFA}\leq}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{\leq} |Q|\big)$$
2. An ABE denoted by $\mathsf{DfaABE}^{>}$ for the relation $R^{\mathsf{DFA}>} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ defined as:
$$R^{\mathsf{DFA}>}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{>} |Q|\big)$$

It is easy to see that given constructions for $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$, we may construct DfaABE simply by running them in parallel. This intuition is formalized in Section 4.8.1.

Then, it suffices to construct the ingredients $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$ – we do so by leveraging existing constructions of *unbounded* kpABE and cpABE for monotone span programs. Since the intuition was discussed in Section 4.1, we directly provide the constructions in Section 4.8.2 and Section 4.8.3 respectively.

4.8.1 Construction: ABE for DFA

Below, we describe the construction of our ABE for DFA formally. We denote our construction as DfaABE.

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

1. Invoke $\mathsf{DfaABE}^{\leq}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{DfaABE}^{>}.\mathsf{Setup}(1^\lambda)$ to obtain $(\mathsf{DfaABE}^{\leq}.\mathsf{mpk}, \mathsf{DfaABE}^{\leq}.\mathsf{msk})$ and $(\mathsf{DfaABE}^{>}.\mathsf{mpk}, \mathsf{DfaABE}^{>}.\mathsf{msk})$ respectively.
2. Output $\mathsf{DfaABE.mpk} = (\mathsf{DfaABE}^{\leq}.\mathsf{mpk}, \mathsf{DfaABE}^{>}.\mathsf{mpk})$ and $\mathsf{DfaABE.msk} = (\mathsf{DfaABE}^{\leq}.\mathsf{msk}, \mathsf{DfaABE}^{>}.\mathsf{msk})$.

$\mathsf{DfaABE.Enc}(\mathsf{DfaABE.mpk}, \mu, \mathbf{x})$: On input the master public key $\mathsf{DfaABE.mpk}$, a message bit μ, and an attribute $\mathbf{x} \in A^{\mathsf{DFA}}$ of unbounded polynomial length (i.e., bounded by 2^λ), do the following:

1. Compute $\mathsf{ct}_1 = \mathsf{DfaABE}^{\leq}.\mathsf{Enc}(\mathsf{DfaABE}^{\leq}.\mathsf{mpk}, \mu, \mathbf{x})$.
2. Compute $\mathsf{ct}_2 = \mathsf{DfaABE}^{>}.\mathsf{Enc}(\mathsf{DfaABE}^{>}.\mathsf{mpk}, \mu, \mathbf{x})$.
3. Output $(\mathsf{ct}_1, \mathsf{ct}_2)$.

$\mathsf{DfaABE.KeyGen}(\mathsf{DfaABE.msk}, \mathsf{DfaABE.mpk}, M)$: On input the master secret key $\mathsf{DfaABE.msk}$, the description of a DFA $M \in B^{\mathsf{DFA}}$ do the following:

1. Compute $\mathsf{sk}_1 = \mathsf{DfaABE}^{\leq}.\mathsf{KeyGen}(\mathsf{DfaABE}^{\leq}.\mathsf{msk}, \mathsf{DfaABE}^{\leq}.\mathsf{mpk}, M)$.
2. Compute $\mathsf{sk}_2 = \mathsf{DfaABE}^{>}.\mathsf{KeyGen}(\mathsf{DfaABE}^{>}.\mathsf{msk}, \mathsf{DfaABE}^{>}.\mathsf{mpk}, M)$.
3. Output $(\mathsf{sk}_1, \mathsf{sk}_2)$.

$\mathsf{DfaABE.Dec}(\mathsf{DfaABE.mpk}, \mathsf{DfaABE.ct}, \mathbf{x}, \mathsf{DfaABE.sk}_M, \mathsf{M})$: On input a ciphertext encoded under attribute $\mathbf{x}$ and a secret key for DFA M, proceed as follows. Let $|Q|$ be the number of states in the machine M.

1. If $|\mathbf{x}| \leq |Q|$, compute $\mu_1 \leftarrow \mathsf{DfaABE}^{\leq}.\mathsf{Dec}(\mathsf{DfaABE}^{\leq}.\mathsf{mpk}, \mathsf{ct}_1, \mathbf{x}, \mathsf{sk}_1, \mathsf{M})$ and output it.
2. If $|\mathbf{x}| > |Q|$, compute $\mu_2 \leftarrow \mathsf{DfaABE}^{>}.\mathsf{Dec}(\mathsf{DfaABE}^{>}.\mathsf{mpk}, \mathsf{ct}_2, \mathbf{x}, \mathsf{sk}_2, \mathsf{M})$ and output it.

Correctness. Correctness follows directly from the correctness of the ingredient schemes $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$, where the former is invoked for the case that $|\mathbf{x}| \leq |Q|$ and the latter otherwise.

Security. Security of the scheme DfaABE follows directly from the security of $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$. In more detail, we have:

Theorem 4.8.1. *Assume that* $\mathsf{DfaABE}^{\leq}$ *and* $\mathsf{DfaABE}^{>}$ *are ABE schemes for relations* $R^{\mathsf{DFA}\leq}$ *and* $R^{\mathsf{DFA}>}$ *respectively, that satisfy selective/selective*/adaptive security. Then,* DfaABE *is an ABE scheme for relation* R^{DFA} *that satisfies selective/selective*/adaptive security.*

The proof is straightforward: for the case that $|\mathbf{x}|\leq|Q|$, the theorem follows from security of $\mathsf{DfaABE}^{\leq}$, otherwise from the security of $\mathsf{DfaABE}^{>}$.

4.8.2 Construction of $\mathsf{DfaABE}^{\leq}$

In this section, we construct the ABE scheme $\mathsf{DfaABE}^{\leq}$ for the relation $R^{\mathsf{DFA}\leq} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ where $R^{\mathsf{DFA}\leq}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{\leq} |Q|\big)$. Our construction is built from the following ingredients:

1. An ABE scheme for the relation $R^{\mathsf{MUKP}} : A^{\mathsf{MUKP}} \times B^{\mathsf{MUKP}} \to \{0,1\}$. Recall from Section 4.7, that $A^{\mathsf{MUKP}} := 2^{\mathbb{Z}}$ is the set of attributes, B^{MUKP} is the set of monotone span programs and $R^{\mathsf{MUKP}}(S, (\mathbf{L}, \rho)) = 1$ iff the span program $(\mathbf{L}, \rho)$ accepts the set $S \in A^{\mathsf{MUKP}}$. We denote such a scheme as kpABE, and construct it in Appendix C.1.2.

2. A map $f_{\mathsf{e}}^{\mathsf{KP}} : A^{\mathsf{DFA}} \to A^{\mathsf{MUKP}}$ and a map $f_{\mathsf{k}}^{\mathsf{KP}} : B^{\mathsf{DFA}} \to B^{\mathsf{MUKP}}$ so that $R^{\mathsf{MUKP}}(S_{\mathbf{x}}, (\mathbf{L}_M, \rho_M)) = 1$ iff $R^{\mathsf{DFA}\leq}(\mathbf{x}, M) = 1$, where $S_{\mathbf{x}} = f_{\mathsf{e}}^{\mathsf{KP}}(\mathbf{x})$ and $(\mathbf{L}_M, \rho_M) = f_{\mathsf{k}}^{\mathsf{KP}}(M)$. These maps are constructed in Section 4.9.1.

The scheme $\mathsf{DfaABE}^{\leq}$ is then defined as follows.

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

1. Invoke $\mathsf{kpABE}.\mathsf{Setup}(1^{\lambda})$ to obtain $(\mathsf{kpABE}.\mathsf{mpk}, \mathsf{kpABE}.\mathsf{msk})$.
2. Output $\mathsf{DfaABE}^{\leq}.\mathsf{mpk} = \mathsf{kpABE}.\mathsf{mpk}$ and $\mathsf{DfaABE}^{\leq}.\mathsf{msk} = \mathsf{kpABE}.\mathsf{msk}$.

$\mathsf{DfaABE}^{\leq}.\mathsf{Enc}(\mathsf{DfaABE}^{\leq}.\mathsf{mpk}, \mu, \mathbf{x})$: On input the master public key $\mathsf{DfaABE}^{\leq}.\mathsf{mpk}$, a message bit μ, and an attribute $\mathbf{x} \in A^{\mathsf{DFA}}$ of unbounded polynomial length (i.e. length at most 2^{λ}), do the following:

1. Convert $\mathbf{x}$ to attribute $S_{\mathbf{x}}$ by computing $S_{\mathbf{x}} = f_{\mathsf{e}}^{\mathsf{KP}}(\mathbf{x})$ as described in Section 4.9.1.
2. Compute $\mathsf{ct} = \mathsf{kpABE}.\mathsf{Enc}(\mathsf{kpABE}.\mathsf{mpk}, \mu, \mathsf{S}_{\mathbf{x}})$ and output it.

$\mathsf{DfaABE}^{\leq}.\mathsf{KeyGen}(\mathsf{DfaABE}^{\leq}.\mathsf{msk}, \mathsf{DfaABE}^{\leq}.\mathsf{mpk}, M)$: On input the master secret key $\mathsf{DfaABE}^{\leq}.\mathsf{msk}$, the description of a DFA $M \in B^{\mathsf{DFA}}$ do the following:

1. Convert M into an MSP $(\mathbf{L}_M, \rho_M)$ by computing $(\mathbf{L}_M, \rho_M) = f_{\mathsf{k}}^{\mathsf{KP}}(M)$ as described in Section 4.9.1.
2. Compute $\mathsf{sk}_M = \mathsf{kpABE}.\mathsf{KeyGen}\big(\mathsf{kpABE}.\mathsf{msk}, \mathsf{kpABE}.\mathsf{mpk}, (\mathbf{L}_M, \rho_M)\big)$ and output it.

$\mathsf{DfaABE}^{\leq}.\mathsf{Dec}(\mathsf{DfaABE}^{\leq}.\mathsf{mpk}, \mathsf{DfaABE}^{\leq}.\mathsf{ct}, \mathbf{x}, \mathsf{DfaABE}^{\leq}.\mathsf{sk}_M, \mathsf{M})$: On input a ciphertext encoded under attribute $\mathbf{x}$ and a secret key for DFA M:

1. Compute $S_{\mathbf{x}} = f_{\mathsf{e}}^{\mathsf{KP}}(\mathbf{x})$ and $(\mathbf{L}_M, \rho_M) = f_{\mathsf{k}}^{\mathsf{KP}}(M)$ as described in Section 4.9.1.
2. Compute $\mu \leftarrow \mathsf{kpABE}.\mathsf{Dec}\big(\mathsf{kpABE}.\mathsf{mpk}, \mathsf{kpABE}.\mathsf{ct}, \mathsf{S}_{\mathbf{x}}, \mathsf{sk}_M, (\mathbf{L}_{\mathsf{M}}, \rho_{\mathsf{M}})\big)$ and output it.

Correctness and Security. Correctness and security follow directly from the "embedding lemma" (Lemma 4.7.1) provided in Section 4.7 by setting

$$A' = A^{\mathsf{DFA}}, \quad B' = B^{\mathsf{DFA}}, \quad R^{F'} = R^{\mathsf{DFA}\leq},$$
$$A = A^{\mathsf{MUKP}}, \quad B = B^{\mathsf{MUKP}}, \quad R^{F} = R^{\mathsf{MUKP}}$$

In more detail, we have the following theorem.

Theorem 4.8.2. *Assume that* kpABE *is an ABE scheme for relation* R^{MUKP} *satisfying selective/selective*/adaptive security. Then,* $\mathsf{DfaABE}^{\leq}$ *is an ABE scheme for relation* $R^{\mathsf{DFA}\leq}$ *satisfying selective/selective*/adaptive security.*

4.8.3 Construction of $\mathsf{DfaABE}^{>}$

In this section, we construct the ABE scheme $\mathsf{DfaABE}^{>}$ for the relation $R^{\mathsf{DFA>}} = \{A^{\mathsf{DFA}} \times B^{\mathsf{DFA}} \to \{0,1\}\}$ where $R^{\mathsf{DFA>}}(\mathbf{x}, M) = M(\mathbf{x}) \wedge \big(|\mathbf{x}| \overset{?}{>} |Q|\big)$. Our construction is built from the following ingredients:

1. An ABE scheme for the relation $R^{\mathsf{MUCP}} : A^{\mathsf{MUCP}} \times B^{\mathsf{MUCP}} \to \{0,1\}$. Recall from Section 2.6, that A^{MUCP} is the set of all monotone span programs, B^{MUCP} is the set of attributes and $R^{\mathsf{MUCP}}((\mathbf{L}, \rho), S) = 1$ iff the span program $(\mathbf{L}, \rho) \in A^{\mathsf{MUCP}}$ accepts the set $S \in B^{\mathsf{MUCP}}$. We denote such a scheme as cpABE, and construct it in Appendix C.1.4.

2. A map $f_{\mathsf{e}}^{\mathsf{CP}} : A^{\mathsf{DFA}} \to A^{\mathsf{MUCP}}$ and a map $f_{\mathsf{k}}^{\mathsf{CP}} : B^{\mathsf{DFA}} \to B^{\mathsf{MUCP}}$ so that $R^{\mathsf{MUCP}}((\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}}), S_M) = 1$ iff $R^{\mathsf{DFA>}}(\mathbf{x}, M) = 1$, where $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}}) = f_{\mathsf{e}}^{\mathsf{CP}}(\mathbf{x})$ and $S_M = f_{\mathsf{k}}^{\mathsf{CP}}(M)$. These maps are constructed in Section 4.9.2.

The scheme $\mathsf{DfaABE}^{>}$ is then defined as follows.

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

1. Invoke $\mathsf{cpABE.Setup}(1^{\lambda})$ to obtain $(\mathsf{cpABE.mpk}, \mathsf{cpABE.msk})$.
2. Output $\mathsf{DfaABE}^{>}.\mathsf{mpk} = \mathsf{cpABE.mpk}$ and $\mathsf{DfaABE}^{>}.\mathsf{msk} = \mathsf{cpABE.msk}$.

$\mathsf{DfaABE}^{>}.\mathsf{Enc}(\mathsf{DfaABE}^{>}.\mathsf{mpk}, \mu, \mathbf{x})$: On input the master public key $\mathsf{DfaABE}^{>}.\mathsf{mpk}$, a message μ, and an attribute $\mathbf{x} \in A^{\mathsf{DFA}}$ of unbounded polynomial length (i.e. length at most 2^{λ}), do the following:

1. Convert $\mathbf{x}$ to MSP $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}})$ by computing $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}}) = f_{\mathsf{e}}^{\mathsf{CP}}(\mathbf{x})$ as described in Section 4.9.2.
2. Compute $\mathsf{ct} = \mathsf{cpABE.Enc}(\mathsf{cpABE.mpk}, \mu, (\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}}))$ and output it.

$\mathsf{DfaABE}^{>}.\mathsf{KeyGen}(\mathsf{DfaABE}^{>}.\mathsf{msk}, \mathsf{DfaABE}^{>}.\mathsf{mpk}, M)$: On input the master secret key $\mathsf{DfaABE}^{>}.\mathsf{msk}$, the description of a DFA M do the following:

1. Convert M into an attribute S_M by computing $S_M = f_{\mathsf{k}}^{\mathsf{CP}}(M)$ as described in Section 4.9.2.
2. Compute $\mathsf{sk} = \mathsf{cpABE.KeyGen}(\mathsf{cpABE.msk}, \mathsf{cpABE.mpk}, S_M)$ and output it.

$\mathsf{DfaABE}^{>}.\mathsf{Dec}(\mathsf{DfaABE}^{>}.\mathsf{mpk}, \mathsf{DfaABE}^{>}.\mathsf{ct}, \mathbf{x}, \mathsf{DfaABE}^{>}.\mathsf{sk}_M, \mathsf{M})$: On input a ciphertext encoded under attribute $\mathbf{x}$ and a secret key sk_M for DFA M:

1. Compute $(\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}}) = f_{\mathsf{e}}^{\mathsf{CP}}(\mathbf{x})$ and $S_M = f_{\mathsf{k}}^{\mathsf{CP}}(M)$ as described in Section 4.9.2.
2. Compute $\mu \leftarrow \mathsf{cpABE.Dec}(\mathsf{cpABE.mpk}, \mathsf{cpABE.ct}, (\mathbf{L}_{\mathbf{x}}, \rho_{\mathbf{x}}), \mathsf{sk}_M, \mathsf{S_M})$ and output it.

Correctness and Security. Correctness and security follow exactly as in Section 4.8.2, by considering the maps defined in Section 4.9.2 instead of Section 4.9.1. In more detail, we have the following theorem:

Theorem 4.8.3. *Assume that* cpABE *is an ABE scheme for relation* R^{MUCP} *satisfying selective/selective*/adaptive security. Then,* $\mathsf{DfaABE}^{>}$ *is an ABE scheme for relation* $R^{\mathsf{DFA}>}$ *satisfying selective/selective*/adaptive security.*

4.9 Mapping DFA Computation to Monotone Span Programs

In this section we will describe how to encode DFA computation over a binary alphabet $\Sigma = \{0, 1\}$ into a monotone span program (MSP). Section 4.9.1 shows the encoding procedure for any DFA machine to a MSP and further how to encode its input to a set of attributes associated with the MSP. In a dual view, Section 4.9.2 shows the encoding procedure for any input string to a MSP while encoding the DFA machine itself as a set of attributes associated with the MSP. For both sections, we denote any DFA machine as $M = (Q, \Sigma, T, q_{\mathsf{st}}, F)$ and $\mathbf{x} \in \Sigma^*$ as its input of arbitrary (polynomial) length.

4.9.1 Encoding DFA to Monotone Span Programs

In this section, we construct two efficiently computable functions (please see Section 4.7 for the notation):

1. $f^{\mathsf{KP}}_{\mathsf{e}} : A^{\mathsf{DFA}} \to A^{\mathsf{MUKP}}$ to encode $\mathbf{w} \in A^{\mathsf{DFA}}$ as a set of attributes $S_{\mathbf{w}} \in A^{\mathsf{MUKP}}$, and
2. $f^{\mathsf{KP}}_{\mathsf{k}} : B^{\mathsf{DFA}} \to B^{\mathsf{MUKP}}$ to encode $M \in B^{\mathsf{DFA}}$ into a MSP $(\mathbf{L}_M, \rho_M) \in B^{\mathsf{MUKP}}$.

We argue that $R^{\mathsf{MUKP}}(S_{\mathbf{w}}, (\mathbf{L}_M, \rho_M)) = 1$ iff $R^{\mathsf{DFA}\leq}(\mathbf{w}, M) = 1$, where $S_{\mathbf{w}} = f^{\mathsf{KP}}_{\mathsf{e}}(\mathbf{w})$ and $(\mathbf{L}_M, \rho_M) = f^{\mathsf{KP}}_{\mathsf{k}}(M)$.

For ease of exposition, we represent the universe of attributes in the following form:

$$A^{\mathsf{MUKP}} := \{\text{“}x_i = b\text{”} \mid i \in [2^\lambda], b \in \{0,1\}\} \cup \{\text{“}\mathsf{String\ length} = i\text{”} \mid i \in [2^\lambda]\} \cup \{\text{“}\mathsf{Dummy}\text{”}\}.$$

We assume that these attributes are embedded into $\mathbb{Z}$ via an injective mapping such as

$$\text{“Dummy”} \mapsto 0, \quad \text{“}x_i = b\text{”} \mapsto 3i + b \quad \text{“String length} = i\text{”} \mapsto 3i + 2.$$

However, for maintaining intuitive notation, we make the mapping implicit. An input string $\mathbf{w} = (w_1, \ldots, w_\ell) \in A^{\mathsf{DFA}}$ of length ℓ is encoded to a set of attributes given by $f_{\mathsf{e}}^{\mathsf{KP}}(\mathbf{w}) = S_{\mathbf{w}} \in A^{\mathsf{MUKP}}$ as:

$$S_{\mathbf{w}} := \{\text{“Dummy”}\} \cup \{\text{“}x_i = w_i\text{”} \mid i \in [\ell]\} \cup \{\text{“String length} = \ell\text{”}\}.$$

When we represent $S_{\mathbf{w}}$ as a set of integers, we have $S_{\mathbf{w}} \subseteq [0, 4\ell]$ and thus in particular, all the values in $S_{\mathbf{w}}$ are bounded by $\mathrm{poly}(\ell)$.

A DFA machine $M = (Q, \Sigma, T, q_{\mathsf{st}}, F) \in B^{\mathsf{DFA}}$ is encoded into a MSP given by $f_{\mathsf{k}}^{\mathsf{KP}}(M) = (\mathbf{L}_M, \rho_M) \in B^{\mathsf{MUKP}}$. Here $\mathbf{L}_M \in \{0, \pm 1\}^{\mathcal{R} \times \mathcal{C}}$ with $\mathcal{R} = 1 + (2 \cdot |Q| + 1) \cdot |Q|$ and $\mathcal{C} = 1 + |Q| + |Q|^2$. The label map ρ_M will be implicit in the description of the matrix $\mathbf{L}_M$. Before providing the construction of $\mathbf{L}_M$, we define the following sub-matrices useful in the construction:

- matrix $\mathbf{I}_Q$ denoting the $|Q| \times |Q|$ identity matrix, and
- matrices $\mathbf{Y}^{(b)} \in \{0, -1\}^{|Q| \times |Q|}, \forall b \in \{0, 1\}$ defined as $\mathbf{Y}^{(b)} := \left[y_{i,j}^{(b)} \right]$ such that:

$$\begin{aligned} y_{i,j}^{(b)} &= -1, \text{ if } T(i, b) = j \text{ (i.e. there is a transition from state } i \text{ to state } j \text{ upon input } b) \\ &= 0, \text{ otherwise} \end{aligned}$$

We also denote $\mathbf{0}_{Q \times Q}$ to be the all-zero matrix of size $|Q| \times |Q|$ and $\mathbf{0}_Q$ as the column-vector of size $|Q|$ containing all 0s.

We define $\mathbf{L}_M$ and the map ρ_M in Table 4.2.

We observe that $\max_i \rho_M(i) \le 4|Q|$, where we regard the attributes as integers through the aforementioned injective mapping. In particular, $\mathbf{L}_M$ is associated with attributes bounded by $\mathrm{poly}(|Q|)$.

The last $|Q|$ rows pertaining to attributes “String length $= i$”, $i \in [|Q|]$ is a $|Q| \times \mathcal{C}$ submatrix containing all zeros except specific locations filled with 1s in a diagonal form as shown. We prove the following theorem.

“Dummy” $\mapsto$	1	-10 $\ldots$ 0	0 $\ldots$ 0	0 $\ldots$ 0	$\ldots$	0 $\ldots$ 0	0 $\ldots$ 0
“$x_1 = 0$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{I}_Q$	$\mathbf{Y}^{(0)}$	$\mathbf{0}_{Q\times Q}$	$\ldots$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$
“$x_1 = 1$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{I}_Q$	$\mathbf{Y}^{(1)}$	$\mathbf{0}_{Q\times Q}$	$\ldots$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$
“$x_2 = 0$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{0}_{Q\times Q}$	$\mathbf{I}_Q$	$\mathbf{Y}^{(0)}$	$\ldots$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$
“$x_2 = 1$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{0}_{Q\times Q}$	$\mathbf{I}_Q$	$\mathbf{Y}^{(1)}$	$\ldots$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
“$x_{\|Q\|} = 0$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$	$\ldots$	$\mathbf{I}_Q$	$\mathbf{Y}^{(0)}$
“$x_{\|Q\|} = 1$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$	$\mathbf{0}_{Q\times Q}$	$\ldots$	$\mathbf{I}_Q$	$\mathbf{Y}^{(1)}$
“String length $= 1$" $\mapsto$	0	0 $\ldots$ 0	0 $\ldots$ 01				
“String length $= 2$" $\mapsto$	0		0 $\ldots$ 00	0 $\ldots$ 01			
$\vdots$	$\vdots$				$\ddots$		
“String length $= \|Q\|$" $\mapsto$	0					0 $\ldots$ 00	0 $\ldots$ 01

Table 4.2: Encoding a DFA M to matrix $\mathbf{L}_M$

Theorem 4.9.1. *Let $\mathbf{L}_{M,\mathbf{w}}$ be the submatrix of $\mathbf{L}_M$ restricted to the rows selected by attribute set $S_\mathbf{w}$ (please see Definition 4.7.1). Then, for any DFA $M = (Q, \Sigma, T, q_{\mathsf{st}}, F) \in B^{\mathsf{DFA}}$ and any input $\mathbf{w} \in A^{\mathsf{DFA}}$ we have $\mathbf{e}_1 \in \mathrm{span}(\mathbf{L}_{M,\mathbf{w}}^\top)$ iff $(M(\mathbf{w}) = 1 \wedge |\mathbf{w}| \leq |Q|)$.*

Proof. We first prove “if" direction. For any $\mathbf{w} \in A^{\mathsf{DFA}}$ with $|\mathbf{w}| = \ell \leq |Q|$, the submatrix $\mathbf{L}_{M,\mathbf{w}}$ of $\mathbf{L}_M$ restricted by $S_\mathbf{w}$ is shown in Table 4.3.

Since M is a DFA, the matrix $\mathbf{Y}^{(b)}$ will always have exactly one “-1" in each of its rows. Let $\mathbf{w} = (w_1, \ldots, w_\ell)$. To prove the theorem, we give an algorithm which constructs a subset of rows $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ of $\mathbf{L}_{M,\mathbf{w}}$ inductively that sums up to $\mathbf{e}_1$ iff $M(\mathbf{w}) = 1$. The algorithm proceeds as follows:

On input $(M, \mathbf{w}, \mathbf{L}_{M,\mathbf{w}})$, it does the following:

1. Initialize $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ with the first row of $\mathbf{L}_{M,\mathbf{w}}$ labelled with attribute “Dummy”.

2. For $i \in [\ell]$, do the following:

"Dummy" $\mapsto$	1	-10 $\ldots$ 0	0 $\ldots$ 0	0 $\ldots$ 0	$\ldots$	0 $\ldots$ 0	0 $\ldots$ 0
"$x_1 = w_1$" $\mapsto$	$\mathbf{0}_Q$	$\mathbf{I}_Q$	$\mathbf{Y}^{(w_1)}$				
"$x_2 = w_2$" $\mapsto$	$\mathbf{0}_Q$		$\mathbf{I}_Q$	$\mathbf{Y}^{(w_2)}$			
$\vdots$	$\vdots$				$\ddots$		
"$x_\ell = w_\ell$" $\mapsto$	$\mathbf{0}_Q$					$\mathbf{I}_Q$	$\mathbf{Y}^{(w_\ell)}$
"String length $= \ell$" $\mapsto$	0					0 $\ldots$ 0	0 $\ldots$ 01

Table 4.3: Submatrix $\mathbf{L}_{M,\mathbf{w}}$ defined by $S_{\mathbf{w}}$ and $\mathbf{L}_M$

(a) If $i = 1$, populate $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ with *second* row of $\mathbf{L}_{M,\mathbf{w}}$ labelled with "$x_1 = w_1$". Discard the remaining $|Q| - 1$ rows in the block labelled with "$x_1 = w_1$".

For the chosen row, let $k_1 \in Q$ be such that $T(1, w_1) = k_1$. By construction this implies $y^{(w_1)}_{1,k_1} = -1$ in $\mathbf{Y}^{(w_1)}$.

(b) If $i \in [2, \ell]$, choose the k_{i-1}-th row in the block labelled with "$x_i = w_i$" and add it to $\widehat{\mathbf{L}}_{M,\mathbf{w}}$. Discard the remaining $|Q| - 1$ rows in the block labelled with "$x_i = w_i$".
For the chosen row, let $k_i \in Q$ be such that $T(k_{i-1}, w_i) = k_i$. By construction this implies $y^{(w_i)}_{k_{i-1},k_i} = -1$ in $\mathbf{Y}^{(w_i)}$.

3. Add the row labelled "String length $= \ell$" to $\widehat{\mathbf{L}}_{M,\mathbf{w}}$. Output $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ and terminate.

It is easy to see that the above algorithm always terminates. The first two rows of $\mathbf{L}_{M,\mathbf{w}}$ labelled with attributes "Dummy" and "$x_1 = w_1$" are chosen in Step 1 and Step $2(a)$ of the above algorithm respectively. The last row is chosen in a natural way in Step 3 based on the length of the input string.

Aside from these, note that the way the remaining rows are added to $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ is governed by the transition function T of the DFA M. Essentially, the computation of $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ mirrors the computation of M on input $\mathbf{w}$. In particular, the *order* in which the rows are selected iteratively in Step 2 always follow a loop invariant: at the end of the i-th iteration the chosen rows sum to a vector $\mathbf{v}_i = (1, 0, \ldots, 0, -1, 0, \ldots, 0)$, where -1 appears exactly at the k_i-th position associated with the $|Q| \times |Q|$-sized block matrix $\mathbf{Y}^{(w_i)}$. Hence, when $M(\mathbf{w}) = 1$ with $|\mathbf{w}| = \ell$, the vectors in $\widehat{\mathbf{L}}_{M,\mathbf{w}}$ at the end of the Step 2 sum to $\mathbf{v}_\ell = (1, 0, \ldots, 0, -1)$. Here -1 is at position $|Q|$ associated with $\mathbf{Y}^{(w_\ell)}$ and is also the final state of M. By construction of $\mathbf{L}_{M,\mathbf{w}}$, it follows that the last row selected in Step 3 labelled with "String length $= \ell$" when added to $\mathbf{v}_\ell$ results to $\mathbf{e}_1$, as intended.

We then prove "only if" direction. For any $\mathbf{w} = (w_1, \ldots, w_\ell) \in \Sigma^\ell$ such that $M(\mathbf{w}) \neq 1$ and $\ell \leq |Q|$, note that the description of $\mathbf{L}_{M,\mathbf{w}}$ forces the first two rows corresponding to attributes "Dummy" and "$x_1 = w_1$" to be chosen to build $\mathbf{e}_1$ progressively. For $i \in [2, \ell - 1]$, let $k_{i-1}, k_i \in Q$ be such that $y^{(w_i)}_{k_{i-1},k_i} = -1$ in $\mathbf{Y}^{(w_i)}$. Consequently, the only choice left for selecting the next row further to nullify the -1 in $y^{(w_i)}_{k_{i-1},k_i}$ is restricted to the k_i-th row in the block labelled with "$x_{i+1} = w_{i+1}$" which again forces the emulation of M's computation on input $\mathbf{w}$. Since $M(\mathbf{w}) \neq 1$, the sum of all the rows at the end of the ℓ-th iteration cannot have a "-1" in its $|Q|^{th}$ position. When added to the row labelled "$\mathsf{String\ length} = \ell$", this does not yield $\mathbf{e}_1$ as desired.

We then consider $\mathbf{w} = (w_1, \ldots, w_\ell) \in \Sigma^\ell$ such that $\ell > |Q|$. In this case, the matrix $\mathbf{L}_{M,\mathbf{w}}$ does not have the last row in Table 4.3. Therefore, we cannot nullify "-1" that appears in the rightmost block as a result of enforced emulation of M's computation. Therefore, we cannot obtain $\mathbf{e}_1$ as desired.

$\square$

4.9.2 Encoding DFA Input Strings to Monotone Span Programs

In this case the DFA machine M is encoded into a set of attributes S_M from an appropriately defined attribute universe while the input string $\mathbf{x} \in \Sigma^*$ will be encoded to a MSP $(\mathbf{L}_\mathbf{x}, \rho_\mathbf{x})$.

We construct two efficiently computable functions:

1. $f^{\mathsf{CP}}_{\mathsf{e}} : A^{\mathsf{DFA}} \to A^{\mathsf{MUCP}}$ to encode $\mathbf{x} \in A^{\mathsf{DFA}}$ into a MSP $(\mathbf{L}_\mathbf{x}, \rho_\mathbf{x}) \in A^{\mathsf{MUCP}}$.

2. $f^{\mathsf{CP}}_{\mathsf{k}} : B^{\mathsf{DFA}} \to B^{\mathsf{MUCP}}$ to encode $M \in B^{\mathsf{DFA}}$ as a set of attributes $S_M \in B^{\mathsf{MUCP}}$.

We argue that $R^{\mathsf{MUCP}}(S_M, (\mathbf{L}_\mathbf{x}, \rho_\mathbf{x})) = 1$ iff $R^{\mathsf{DFA}>}(\mathbf{x}, M) = 1$, where $S_M = f^{\mathsf{CP}}_{\mathsf{k}}(M)$ and $(\mathbf{L}_\mathbf{x}, \rho_\mathbf{x}) = f^{\mathsf{CP}}_{\mathsf{e}}(\mathbf{x})$.

For ease of exposition, we represent the universe of attributes as follows:

$$B^{\mathsf{MUCP}} := \{(b, i, j) \mid b \in \{0,1\}, i, j \in [2^\lambda]\} \cup \{\text{"}\mathsf{Size} = \mathsf{s}\text{"} \mid \mathsf{s} \in [2^\lambda]\} \cup \{\text{"}\mathsf{Dummy}\text{"}\}.$$

We assume that these attributes are embedded into $\mathbb{Z}$ via an injective mapping such as

$$\text{“Dummy”} \mapsto 0, \quad \text{“}(b,i,j)\text{”} \mapsto 4((i+j)^2+j)+2b \quad \text{“Size} = \text{s”} \mapsto 2\text{s}+1,$$

But for maintaining intuitive notation, we make the mapping implicit.

A DFA $M = (Q, \Sigma, T, q_{\mathsf{st}}, F) \in B^{\mathsf{DFA}}$ is encoded as a set of attributes given by $f_{\mathsf{k}}^{\mathsf{CP}}(M) = S_M \in B^{\mathsf{MUCP}}$ as:

$$S_M := \{\text{“Dummy”}\} \cup \{(b,i,j) \in \Sigma \times Q^2 \mid T(i,b) = j\} \cup \{\text{“Size} = |Q|\text{”}\}.$$

When we represent S_M as a set of integers, we have $S_M \subseteq [0, 20|Q|^2]$ and thus in particular, all the values in S_M are bounded by $\mathrm{poly}(|Q|)$.

An input string $\mathbf{x} = (x_1, \ldots, x_\ell) \in A^{\mathsf{DFA}}$ of length ℓ is encoded into a MSP given by $f_{\mathsf{e}}^{\mathsf{CP}}(\mathbf{x}) = (\mathbf{L}_\mathbf{x}, \rho_\mathbf{x}) \in A^{\mathsf{MUCP}}$. Here $\mathbf{L}_\mathbf{x} \in \{0, \pm 1\}^{\mathcal{R}\times\mathcal{C}}$ with $\mathcal{R} = 1 + \ell^3 + \ell$ and $\mathcal{C} = 1 + \ell + \ell^2$. The label map $\rho_\mathbf{x}$ will be implicit in the description of the matrix $\mathbf{L}_\mathbf{x}$. Before providing the construction of $\mathbf{L}_\mathbf{x}$, we define the following sub-matrices useful in the construction:

- matrix $\mathbf{I}_\ell$ denoting the $\ell \times \ell$ identity matrix and a column-vector $\mathbf{g}_\ell = \underbrace{(1, \ldots, 1)}_{\ell}^\top$

- matrices $\mathbf{S}_\ell$ and $\mathbf{T}_\ell$ such that

$$\mathbf{S}_\ell := \mathbf{I}_\ell \otimes \mathbf{g}_\ell = \begin{bmatrix} \mathbf{g}_\ell & \mathbf{0}_\ell & \ldots & \mathbf{0}_\ell \\ \mathbf{0}_\ell & \mathbf{g}_\ell & \ldots & \mathbf{0}_\ell \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_\ell & \mathbf{0}_\ell & \ldots & \mathbf{g}_\ell \end{bmatrix}_{\ell^2\times\ell} \text{, where } \mathbf{0}_\ell \text{ is the all-zero column-vector of size } \ell$$

  and $\mathbf{T}_\ell = -\mathbf{g}_\ell \otimes \mathbf{I}_\ell = [-\mathbf{I}_\ell \| \ldots \| - \mathbf{I}_\ell]^\top$ of size $\ell^2 \times \ell$.

For a fixed $b \in \{0,1\}$, we say “*associate* $[\mathbf{S}_\ell \| \mathbf{T}_\ell]$ *with* b"[4] when we label the rows of $[\mathbf{S}_\ell \| \mathbf{T}_\ell]$ as shown in Table 4.4.

We also denote $\mathbf{0}_{\ell^2}$, $\mathbf{0}_{\ell^2\times\ell}$ and $\mathbf{0}_{\ell\times\ell}$ to be all-zero column-vector of size ℓ^2 and all-zero matrices of size $\ell^2 \times \ell$ and $\ell \times \ell$ respectively. We now define $\mathbf{L}_\mathbf{x}$ with its rows labelled with attributes as specified in Table 4.5.

[4]For brevity, we express this as $b \Leftrightarrow [\mathbf{S}_\ell \| \mathbf{T}_\ell]$ in the final description of $\mathbf{L}_\mathbf{x}$.

$$\begin{array}{r|cccc|c|}
(b,1,1)\mapsto & & & & & \\
\vdots & \mathbf{g}_\ell & \mathbf{0}^\ell & \cdots & \mathbf{0}^\ell & -\mathbf{I}_\ell \\
(b,1,\ell)\mapsto & & & & & \\
(b,2,\ell)\mapsto & & & & & \\
\vdots & \mathbf{0}^\ell & \mathbf{g}_\ell & \cdots & \mathbf{0}^\ell & -\mathbf{I}_\ell \\
(b,2,\ell)\mapsto & & & & & \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
(b,\ell,1)\mapsto & & & & & \\
\vdots & \mathbf{0}^\ell & \mathbf{0}^\ell & \cdots & \mathbf{g}_\ell & -\mathbf{I}_\ell \\
(b,\ell,\ell)\mapsto & & & & &
\end{array}$$

Table 4.4: Submatrix $[\mathbf{S}_\ell \| \mathbf{T}_\ell]$ with its row label map

We observe that we have $\max_i \rho_{\mathbf{x}}(i) \leq 20\ell^2$, where we regard the attributes as integers through the aforementioned injective mapping. In particular, $\mathbf{L}_{\mathbf{x}}$ is associated with attributes bounded by $\mathrm{poly}(\ell)$.

"Dummy" $\mapsto$	1	-10…0	0…0	0…0	…	0…0	0…0
$x_1 \Leftrightarrow$	$\mathbf{0}_{\ell^2}$	$\mathbf{S}_\ell$	$\mathbf{T}_\ell$	$\mathbf{0}_{\ell^2\times\ell}$	…	$\mathbf{0}_{\ell^2\times\ell}$	$\mathbf{0}_{\ell^2\times\ell}$
$x_2 \Leftrightarrow$	$\mathbf{0}_{\ell^2}$	$\mathbf{0}_{\ell^2\times\ell}$	$\mathbf{S}_\ell$	$\mathbf{T}_\ell$	…	$\mathbf{0}_{\ell^2\times\ell}$	$\mathbf{0}_{\ell^2\times\ell}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_\ell \Leftrightarrow$	$\mathbf{0}_{\ell^2}$	$\mathbf{0}_{\ell^2\times\ell}$	$\mathbf{0}_{\ell^2\times\ell}$	$\mathbf{0}_{\ell^2\times\ell}$	…	$\mathbf{S}_\ell$	$\mathbf{T}_\ell$
"Size $= 1$" $\mapsto$	0						
$\vdots$	$\vdots$	$\mathbf{0}_{\ell\times\ell}$	$\mathbf{0}_{\ell\times\ell}$	$\mathbf{0}_{\ell\times\ell}$	…	$\mathbf{0}_{\ell\times\ell}$	$\mathbf{I}_\ell$
"Size $= \ell$" $\mapsto$	0						

Table 4.5: Encoding a string $\mathbf{x}$ to matrix $\mathbf{L}_{\mathbf{x}}$

The last ℓ rows pertaining to attributes "Size $= i$", $i \in [\ell]$ is a $\ell \times \mathcal{C}$ submatrix containing all zeros except an identity matrix block $\mathbf{I}_\ell$ located under the rightmost $\mathbf{T}_\ell$ with its i-th row labelled with attribute "Size $= i$", $\forall i \in [\ell]$. We show the following.

Theorem 4.9.2. *Let $\mathbf{L}_{M,\mathbf{x}}$ be the submatrix of $\mathbf{L}_{\mathbf{x}}$ restricted to the rows selected by the set S_M (please see Definition 4.7.1). Then, for any DFA $M = (Q, \Sigma, T, q_{\mathsf{st}}, F) \in B^{\mathsf{DFA}}$ and any input $\mathbf{x} \in A^{\mathsf{DFA}}$ we have $\mathbf{e}_1 \in \mathrm{span}(\mathbf{L}_{M,\mathbf{x}}^\top)$ iff $\big(M(\mathbf{x}) = 1 \wedge |\mathbf{x}| \geq |Q|\big)$.*

Proof. We first remove all the all-zero columns from $\mathbf{L}_{M,\mathbf{x}}$ and call the remaining matrix as $\mathbf{L}_{M,\mathbf{x}}$ w.l.o.g. since these columns do not influence on whether $\mathbf{e}_1 \in \mathrm{span}(\mathbf{L}_{M,\mathbf{x}}^\top)$ or not. This simplification ensures that $\mathbf{L}_{M,\mathbf{x}}$ is given as shown in Table 4.6. Note that the rows present in $\mathbf{L}_{M,\mathbf{x}}$ is governed by the transition function, T of M (via the row labels in $\mathbf{L}_{\mathbf{x}}$). We also note that the last row in Table 4.6 will be missing if we have $|\mathbf{x}| < |Q|$. Therefore, the matrix $\mathbf{Y}^{(b)}$ here is the same as that was defined in Section 4.9.1. Hence,

"Dummy" $\mapsto$	1	-10...0	0...0	0...0	...	0...0	0...0
$x_1 \Leftrightarrow$	$\mathbf{0}_Q$	$\mathbf{I}_Q$	$\mathbf{Y}^{(x_1)}$				
$x_2 \Leftrightarrow$	$\mathbf{0}_Q$		$\mathbf{I}_Q$	$\mathbf{Y}^{(x_2)}$			
$\vdots$	$\vdots$				$\ddots$		
$x_\ell \Leftrightarrow$	$\mathbf{0}_Q$					$\mathbf{I}_Q$	$\mathbf{Y}^{(x_\ell)}$
"Size = $\|Q\|$" $\mapsto$	0					0...0	0...01

Table 4.6: Submatrix $\mathbf{L}_{M,\mathbf{x}}$ defined by S_M and $\mathbf{L}_{\mathbf{x}}$

the proof follows identically to that of Theorem 4.9.1. $\square$

4.10 Putting it all together: ABE for DFA

In this section, we discuss instantiation of our generic construction of ABE for DFA by putting together all the ingredients developed so far.

As we have seen in Sec. 4.8.1, ABE for R^{DFA} (i.e., ABE for DFA) can be constructed from ABE for $R^{\mathsf{DFA}\geq}$ and ABE for $R^{\mathsf{DFA}\leq}$. Furthermore, as we have seen in Theorem 4.8.3 (resp., Theorem 4.8.2), ABE for $R^{\mathsf{DFA}>}$ (resp., ABE for $R^{\mathsf{DFA}\leq}$) is implied by ABE for R^{MUCP} (resp., R^{MUKP}).

To instantiate the ABE for R^{MUKP}, we use the construction in Appendix C.1.2. As was shown in Theorem C.1.5, this construction is semi-adaptively secure under the MDDH_k assumption. To instantiate the ABE for R^{MUCP}, we use the construction in Appendix C.1.4. As was shown in Theorem C.1.15, this construction satisfies selective* security under the DLIN assumption. Putting all pieces together, we obtain the following theorem.

Theorem 4.10.1. *There exists selective* secure key-policy ABE for R^{DFA} from the* DLIN *assumption.*

Ciphertext Policy ABE for DFA. We observe that our construction DfaABE uses the underlying kpABE and cpABE in a symmetric way. Thus, by swapping the use of kpABE and cpABE in our construction, we can equivalently construct ciphertext-policy ABE for DFA. Recall that analogous to ABE for MSP (Section 4.7), the ciphertext-policy variant of ABE for DFA is defined simply by swapping the order of the domains in the relation R^{DFA}. In more detail, we set $A^{\mathsf{CPDFA}} = B^{\mathsf{DFA}}$ and $B^{\mathsf{CPDFA}} = A^{\mathsf{DFA}}$ and define the relation R^{CPDFA} analogously for a ciphertext policy scheme for DFA. Thus, in a ciphertext-policy scheme, the encryptor to encrypt a machine and the key generator to compute a key for an input $\mathbf{x}$.

To modify DfaABE to be ciphertext-policy, we exchange the maps used by KeyGen and Enc in the constructions of $\mathsf{DfaABE}^{\leq}$ and $\mathsf{DfaABE}^{>}$ in Sections 4.8.2 and 4.8.3 respectively. For instance, to construct a ciphertext-policy variant of $\mathsf{DfaABE}^{\leq}$, we modify the encrypt and key generation algorithms so that:

1. The key generation algorithm receives as input an attribute $\mathbf{x}$, converts it to attributes $S_{\mathbf{x}}$ using the map defined in Section 4.9.1 and computes cpABE key for $S_{\mathbf{x}}$.
2. The encryption algorithm receives as input an MSP M, converts it to an MSP $(\mathbf{L}_M, \rho_M)$ using the map defined in Section 4.9.1 and computes cpABE encryption for policy $(\mathbf{L}_M, \rho_M)$.

The modification to $\mathsf{DfaABE}^{>}$ is analogous. The compiler DfaABE remains the same.

Thus, we additionally obtain the following theorem:

Theorem 4.10.2. *There exists selective* secure ciphertext-policy ABE for R^{DFA} from the* DLIN *assumption.*

CHAPTER 5

Conclusions

In this thesis, we studied computation on encrypted data, where computation is modeled using uniform models of computation.

We provided a generic approach to compile FE for Turing machines, given the same for circuits. This enables translating advances in the FE for circuits literature directly to the FE for Turing machine literature and reduces the underlying assumptions to be polynomial rather than sub-exponential. We also obtained such generic compilers for multi-input FE and iO. Several interesting questions arise from our work. For instance, it would be useful to study if our techniques can be extended to support *unbounded* number of encryptors in the multi-input setting. It would also be desirable to replace the single input public key FE for circuits in the multi-input construction by a private key scheme since the multi-input construction is in the private-key setting. Finally, we believe our techniques can find other applications in replacing the use of iO by FE. We leave these to future work.

Continuing with the same motivation, we provided two new constructions of ABE for finite automata based on static assumptions from lattices and bilinear maps. The first one shows how to construct ABE schemes (and also its generalizations namely, predicate encryption, bounded key FE and reusable garbling) for nondeterministic finite automata from the learning with errors assumption. An important question here is whether we may generalize our techniques to support more advanced models of computation like Turing machines or RAM. In this context we note that getting an ABE scheme for Turing machines does not really imply an ABE scheme for RAMs with all efficiency guarantees. A RAM program, by definition, has random access to an additional database much larger than the local memory available to the program. Therefore, the runtime of such ABE schemes supporting RAM may be at most sub-linear in the database size. On the contrary, the runtime for an ABE scheme for Turing machines may be at least linear in the size of the input since it has to parse the whole input at least. While Turing machines and RAMs are equivalent in terms of computational power, the efficiency properties and meaningfulness of a RAM are lost when it is converted to a Turing machine. Another important point to note is that this construction is restricted to the private-key setting. It would be interesting to design a public-key variant of our scheme as well as to upgrade the security proof to satisfy adaptive security. In the second construction, we obtained both key-policy and ciphertext-policy, public key ABE schemes for deterministic finite automata, given the same for monotone span programs. Our construction is generic and modular and is the first to be based on standard static assumptions over bilinear maps. It also reduces the question of obtaining adaptive security for deterministic finite automata for both key-policy and ciphertext-policy ABE to that of adaptive security for unbounded key-policy and ciphertext-policy ABE for monotone span programs. Two interesting questions arise from this work. The first one is whether our techniques can be further generalized to also support nondeterministic finite automata. Secondly, obtaining adaptively secure unbounded ABE schemes for monotone span programs is also an interesting direction to pursue.

Finally, all our results employ new techniques. Hence, it would be nice to find other applications for them.

Last but not least, our constructions in this thesis are feasibility results and mainly of theoretical interest. Therefore, in general, another important future direction includes designing ABE and FE systems that are more efficient and useful in practice.

APPENDIX A

Appendices for Chapter 2

A.1 Missing Details in Proof of Theorem 2.7.1

Proof. In the following we argue that consecutive hybrids as defined in Section 2.7.3 are indistinguishable.

Claim A.1.1. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(0)$ *and* $\mathcal{H}(1,1)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(0)$ and $\mathcal{H}(1,1)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_1$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE}_1.\mathsf{PK}$ from the $\mathsf{1FE}_1$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and two random strings $\mathsf{ct}_1, \mathsf{ct}_2 \leftarrow \mathcal{C}^{\mathsf{SKE}}$, where $\mathcal{C}^{\mathsf{SKE}}$ denotes the ciphertext space of the SKE scheme.
2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
   (b) $\mathcal{B}$ executes the oblivious TM M on $\mathbf{w}_0$ to learn the (symbol, state) pairs $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$ and $(\sigma_{T-2}^0, \mathsf{q}_{T-2}^0)$ at time steps $T-1$ and $T-2$ respectively. It also records the time steps $(T', T-2)$ and $(T'', T-3)$ when the individual components of these two (symbol, state) pairs are generated and then prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}_0^\ell, \widehat{\mathcal{D}}_1^\ell)$ for the $\mathsf{1FE}_1$ challenger as follows
      i. For $b = 0$, $\widehat{\mathcal{D}}_0^\ell = \{\mathbf{x}_0 = (x_{0,1}, \ldots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^0)$, with $\mathsf{Trap}^0.\mathsf{mode\text{-}real} = 1$ and all other fields set to $\perp$.

$\mathsf{mode\text{-}real} : \perp$	$\mathsf{key\text{-}id} : \mathsf{salt}$	$\mathsf{val}_0 : w_{0,i}$	$\mathsf{val}_1 : w_{1,i}$	$\mathsf{SKE.K} : \perp$	$\perp$
$\mathsf{mode\text{-}trap}_1 : 1$	$\mathsf{Target\ TS}_1 : T-1$	$\mathsf{Sym\ TS}_1 : T'$	$\mathsf{Sym\ val}_1 : \sigma^0_{T-1}$	$\mathsf{ST\ TS}_1 : T-2$	$\mathsf{ST\ val}_1 : \mathsf{q}^0_{T-1}$
$\mathsf{mode\text{-}trap}_2 : 1$	$\mathsf{Target\ TS}_2 : T-2$	$\mathsf{Sym\ TS}_2 : T''$	$\mathsf{Sym\ val}_2 : \sigma^0_{T-2}$	$\mathsf{ST\ TS}_2 : T-3$	$\mathsf{ST\ val}_2 : \mathsf{q}^0_{T-2}$
$\mathsf{mode\text{-}trap}_3 : \perp$	$\mathsf{Target\ TS} : \perp$	$\mathsf{Sym\ TS} : \perp$	$\perp$	$\mathsf{ST\ TS} : \perp$	$\perp$

Figure A.1: Trap^1 configuration in $\mathcal{H}(1,1)$

ii. For $b = 1$, $\widehat{\mathcal{D}}^\ell_1 = \{\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with the modified fields in Trap^1 as shown in Figure A.1.

(c) It sends the distribution pair to the $\mathsf{1FE}_1$ challenger and relays the response back to $\mathcal{A}$.

(d) To simulate a function key for M, $\mathcal{B}$ first requests for a function key to the $\mathsf{1FE}_1$ challenger for the function $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$ and receives $\mathsf{SK}_{\mathsf{ReRand}}$. $\mathcal{B}$ computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ by itself and returns to $\mathcal{A}$ a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Observe that for all time steps $t \notin \{T', T-2, T'', T-3\}$, the decryption outputs are exactly the same ciphertexts in both the $\mathcal{H}(0)$ and $\mathcal{H}(1,1)$, since these ciphertexts are computed according to the real world functionality of Next. At a time step $t \in \{T', T-2, T'', T-3\}$ in $\mathcal{H}(1,1)$, the decryption *mimics* the real world decryption of $\mathcal{H}(0)$ due to the execution paths in the Next function conditioned on $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_1 = \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_2 = 1$. Therefore, $\mathcal{B}$ is an admissible adversary against the $\mathsf{1FE}_1$ challenger since the outputs for the two challenge message sets are exactly the same. If $b = 0$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(0)$, while if $b = 1$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(1,1)$. Thus the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. □

Claim A.1.2. *If* SKE *is a secure symmetric-key encryption scheme, then hybrids* $\mathcal{H}(1,1)$ *and* $\mathcal{H}(1,2)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(1,1)$ and $\mathcal{H}(1,2)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the SKE scheme as follows.

1. $\mathcal{B}$ samples $(\mathsf{1FE}_1.\mathsf{PK}, \mathsf{1FE}_1.\mathsf{MSK}) \leftarrow \mathsf{1FE}_1.\mathsf{Setup}(1^\lambda)$, $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda)$ and $\mathsf{salt} \leftarrow \{0,1\}^\lambda$. It sends $\mathsf{PK} = \mathsf{1FE}_1.\mathsf{PK}$ to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}^\ell_b$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.

(a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \dots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.

(b) $\mathcal{B}$ executes the oblivious TM M on $\mathbf{w}_0$ to learn the (symbol, state) pairs $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$ and $(\sigma_{T-2}^0, \mathsf{q}_{T-2}^0)$ at time steps $T-1$ and $T-2$ respectively. It also records the time steps $(T', T-2)$ and $(T'', T-3)$ when the individual components of these two (symbol, state) pairs are generated. It then simulates the encryption oracle by computing $\mathsf{CT}_i = \mathsf{1FE_1.Enc}(\mathsf{1FE_1.PK}, x_{1,i})$, where $\forall i \in [\ell], x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.1. It returns the ciphertext $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[\ell]}$ to $\mathcal{A}$.

(c) To simulate a function key for M, $\mathcal{B}$ does the following.

i. It computes $\mathsf{SK}_{\mathsf{ReRand}} \leftarrow \mathsf{1FE_1.KeyGen}(\mathsf{1FE_1.MSK}, \mathsf{ReRand}_{\mathsf{1FE_2.PK},\mathsf{salt},\mathsf{q_{st}},\perp,\perp})$.

ii. It then computes $\mathsf{1FE_2}$ encodings of $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$ as follows.

- Compute a delegated cPRF key $\mathsf{K}_T = \mathsf{F.KeyDel}(\mathsf{K}_0, f_T)$ and generate the encryption randomness for time step $T-1$ as $r_{T-1} = \mathsf{F.Eval}(\mathsf{K}_0, (T-1\|\mathsf{salt}))$.

- Compute the $\mathsf{1FE_2}$ *symbol* ciphertext to be given as output at time step T' for the future time step $T-1$ as $\mathsf{CT}_{\mathsf{sym},T-1}^0 = \mathsf{1FE_2.Enc}(\mathsf{1FE_2.PK}_1, \mathbf{z}_1^0; r_{T-1})$, where $\mathbf{z}_1^0 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_T, T-1, \ell, \sigma_{T-1}^0, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.1.

- Compute the $\mathsf{1FE_2}$ *state* ciphertext to be given as output at time step $T-2$ for the future time step $T-1$ as $\mathsf{CT}_{\mathsf{st},T-1}^0 = \mathsf{1FE_2.Enc}(\mathsf{1FE_2.PK}_2, \mathbf{z}_2^0; r_{T-1})$, where $\mathbf{z}_2^0 = (\mathsf{ST}, \mathsf{q}_{T-1}^0)$.

iii. It sends the $\mathsf{1FE_2}$ ciphertexts $\mathsf{CT}_{\mathsf{sym},T-1}^0, \mathsf{CT}_{\mathsf{st},T-1}^0$ to the challenger of the SKE scheme and gets back $\mathsf{ct}_1, \mathsf{ct}_2$.

iv. $\mathcal{B}$ then computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE_2.KeyGen}(\mathsf{1FE_2.MSK}, \mathsf{Next}_{\mathsf{1FE_2.PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the only difference between the two hybrids is that the SKE encryptions programmed in the function key is random in $\mathcal{H}(1,1)$ and are valid SKE encryptions of $(\mathsf{CT}_{\mathsf{sym},T-1}^0, \mathsf{CT}_{\mathsf{st},T-1}^0)$ encoding the (symbol, state) pair for time step $T-1$ in $\mathcal{H}(1,2)$. Hence the advantage of an adversary who distinguishes between the two hybrids translates to an advantage of an adversary against the SKE scheme. $\square$

Claim A.1.3. *If* $\mathsf{1FE_1}$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(1,2)$ *and* $\mathcal{H}(1,3)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(1,2)$ and $(1,3)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE_1}$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE_1.PK}$ from the $\mathsf{1FE_1}$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE_2.PK}, \mathsf{1FE_2.MSK}) \leftarrow \mathsf{1FE_2.Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and a key $\mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
   (b) $\mathcal{B}$ executes the oblivious TM M on $\mathbf{w}_0$ to learn the (symbol, state) pairs $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$ and $(\sigma_{T-2}^0, \mathsf{q}_{T-2}^0)$ at time steps $T-1$ and $T-2$ respectively. It also records the time steps $(T', T-2)$ and $(T'', T-3)$ when the individual components of these two (symbol, state) pairs are generated and then prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}_0^\ell, \widehat{\mathcal{D}}_1^\ell)$ for the $\mathsf{1FE_1}$ challenger as follows.
      i. For $b = 0$, $\widehat{\mathcal{D}}_0^\ell = \{\mathbf{x}_0 = (x_{0,1}, \ldots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^0)$ with the fields of Trap^0 being same as that of in Trap^1 in $\mathcal{H}(1,2)$ as per Figure A.1.
      ii. For $b = 1$, $\widehat{\mathcal{D}}_1^\ell = \{\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with the modified fields in Trap^1 as shown in Figure A.2.

mode-real : $\perp$	key-id : salt	val_0 : $w_{0,i}$	val_1 : $w_{1,i}$	SKE.K : K	$\perp$
$\mathsf{mode\text{-}trap}_1$: $\perp$	Target TS_1 : $\perp$	Sym TS_1 : $\perp$	Sym val_1 : $\perp$	ST TS_1 : $\perp$	ST val_1 : $\perp$
$\mathsf{mode\text{-}trap}_2$: 1	Target TS_2 : $T-2$	Sym TS_2 : T''	Sym val_2 : σ_{T-2}^0	ST TS_2 : $T-3$	ST val_2 : q_{T-2}^0
$\mathsf{mode\text{-}trap}_3$: 1	Target TS : $T-1$	Sym TS : T'	$\perp$	ST TS : $T-2$	$\perp$

Figure A.2: Trap^1 configuration in $\mathcal{H}(1,3)$

   (c) It sends the distribution pair to the $\mathsf{1FE_1}$ challenger and relays the response back to $\mathcal{A}$.
   (d) To simulate a function key for M, $\mathcal{B}$ does the following.
      i. It requests for a function key for $\mathsf{ReRand}_{\mathsf{1FE_2.PK},\mathsf{salt},\mathsf{q_{st}},\perp,\perp}$ to the $\mathsf{1FE_1}$ challenger and receives $\mathsf{SK}_{\mathsf{ReRand}}$.
      ii. It then computes $\mathsf{1FE_2}$ encodings of $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$ as follows.
         - Compute a delegated cPRF key $\mathsf{K}_T = \mathsf{F.KeyDel}(\mathsf{K}_0, f_T)$ and generate the encryption randomness for time step $T-1$ as $r_{T-1} = \mathsf{F.Eval}(\mathsf{K}_0, (T-1\|\mathsf{salt}))$.
         - Compute the $\mathsf{1FE_2}$ *symbol* ciphertext to be given as output at time step T' for the future time step $T-1$ as $\mathsf{CT}_{\mathsf{sym},T-1}^0 = \mathsf{1FE_2.Enc}(\mathsf{1FE_2.PK_1}, \mathbf{z}_1^0; r_{T-1})$, where $\mathbf{z}_1^0 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_T, T-1, \ell, \sigma_{T-1}^0, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.2 now.

- Compute the $\mathsf{1FE}_2$ *state* ciphertext to be given as output at time step $T-2$ for the future time step $T-1$ as $\mathsf{CT}^0_{\mathsf{st},T-1} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_2, \mathbf{z}^0_2; r_{T-1})$, where $\mathbf{z}^0_2 = (\mathsf{ST}, \mathsf{q}^0_{T-1})$.

iii. Once it has generated the two $\mathsf{1FE}_2$ ciphertexts $\mathsf{CT}^0_{\mathsf{sym},T-1}$ and $\mathsf{CT}^0_{\mathsf{st},T-1}$, it computes two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE}.\mathsf{Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{sym},T-1})$ and $\mathsf{ct}_2 = \mathsf{SKE}.\mathsf{Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{st},T-1})$.

iv. Finally, it computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Observe that for all time steps $t \notin \{T', T-2\}$, the decryption outputs are exactly the same ciphertexts in both $\mathcal{H}(1,2)$ and $\mathcal{H}(1,3)$. At time step $t \in \{T', T-2\}$ in $\mathcal{H}(1,2)$, $\mathsf{Trap}^0.\mathsf{mode\text{-}trap}_1 = 1$ (in Figure A.1) dictates the decryption to output two decomposed components of a single $\mathsf{1FE}_2$ ciphertext, one component encoding $\mathsf{Trap}^0.\mathsf{Sym}\ \mathsf{val}_1 = \sigma^0_{T-1}$ at time step T' and the other encoding $\mathsf{Trap}^0.\mathsf{ST}\ \mathsf{val}_1 = \mathsf{q}^0_{T-1}$ at time step $T-2$. Alternatively in $\mathcal{H}(1,3)$, $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = 1$ (in Figure A.2) dictates the decryption to firstly use $\mathsf{Trap}^1.\mathsf{SKE}.\mathsf{K} = \mathsf{K}$ to decrypt the hardwired ciphertext ct_1 and output $\mathsf{CT}^0_{\mathsf{sym},T-1}$ at time step T' (respectively, ct_2 and output $\mathsf{CT}^0_{\mathsf{st},T-1}$ at time step $T-2$). In both the hybrids, these symbol and state ciphertext pieces are computed for target time step $T-1$. Thus $\mathcal{B}$ is an admissible $\mathsf{1FE}_1$ adversary. If $b=0$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(1,2)$, while if $b=1$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(1,3)$. Hence the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. □

Claim A.1.4. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(1,3)$ *and* $\mathcal{H}(1,4)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(1,3)$ and $\mathcal{H}(1,4)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_1$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE}_1.\mathsf{PK}$ from the $\mathsf{1FE}_1$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and a key $\mathsf{K} \leftarrow \mathsf{SKE}.\mathsf{KeyGen}(1^\lambda)$.
2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}^\ell_b$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F}.\mathsf{Setup}(1^\lambda)$.

(b) $\mathcal{B}$ executes the oblivious TM M on $\mathbf{w}_0$ to learn the (symbol, state) pairs $(\sigma^0_{T-1}, \mathsf{q}^0_{T-1})$ and $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ at time steps $T-1$ and $T-2$ respectively. It also records the time steps $(T', T-2)$ and $(T'', T-3)$ when the individual components of these two (symbol, state) pairs are generated. It then computes a root key punctured at point $(T-1\|\mathsf{salt})$ as $\mathsf{K}_0^{T-1} = \mathsf{F.Constrain}(\mathsf{K}_0, (T-1\|\mathsf{salt}))$ and then prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}^\ell_0, \widehat{\mathcal{D}}^\ell_1)$ for the $\mathsf{1FE}_1$ challenger as follows.

   i. For $b = 0$, $\widehat{\mathcal{D}}^\ell_0 = \{\mathbf{x}_0 = (x_{0,1}, \dots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ with the fields of Trap^1 being same as that of in Trap^1 in $\mathcal{H}(1,3)$ as per Figure A.2.

   ii. For $b = 1$, $\widehat{\mathcal{D}}^\ell_1 = \{\mathbf{x}_1 = (x_{1,1}, \dots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0^{T-1}, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with Trap^1 as per Figure A.2.

(c) It sends the distribution pair to the $\mathsf{1FE}_1$ challenger and relays the response back to $\mathcal{A}$.

(d) To simulate a function key for M, $\mathcal{B}$ does the following.

   i. It requests for a function key for $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$ to the $\mathsf{1FE}_1$ challenger and receives $\mathsf{SK}_{\mathsf{ReRand}}$.

   ii. It then computes $\mathsf{1FE}_2$ encodings of $(\sigma^0_{T-1}, \mathsf{q}^0_{T-1})$, as follows.

      - Compute a punctured, delegated key $\mathsf{K}_T^{T-1} = \mathsf{F.KeyDel}(\mathsf{K}_0^{T-1}, f_T)$ and generate the encryption randomness for time step $T-1$ as $r_{T-1} = \mathsf{F.Eval}(\mathsf{K}_0, (T-1\|\mathsf{salt}))$.
      - Compute the $\mathsf{1FE}_2$ *symbol* ciphertext to be given as output at time step T' for the future time step $T-1$ as $\mathsf{CT}^0_{\mathsf{sym},T-1} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_1, \mathbf{z}^0_1; r_{T-1})$, where $\mathbf{z}^0_1 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_T^{T-1}, T-1, \ell, \sigma^0_{T-1}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.2.
      - Compute the $\mathsf{1FE}_2$ *state* ciphertext to be given as output at time step $T-2$ for future time step $T-1$ as $\mathsf{CT}^0_{\mathsf{st},T-1} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_2, \mathbf{z}^0_2; r_{T-1})$, where $\mathbf{z}^0_2 = (\mathsf{ST}, \mathsf{q}^0_{T-1})$.

   iii. Once it has generated the two $\mathsf{1FE}_2$ ciphertexts $\mathsf{CT}^0_{\mathsf{sym},T-1}$ and $\mathsf{CT}^0_{\mathsf{st},T-1}$, it computes two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{sym},T-1})$ and $\mathsf{ct}_2 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{st},T-1})$.

   iv. Finally, it computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the only difference in $\mathcal{H}(1,3)$ and $\mathcal{H}(1,4)$ is the replacement of the root cPRF key K_0 with a punctured root key K_0^{T-1} at point $(T-1\|\mathsf{salt})$ in time step $T-1$ in the $\mathsf{1FE}_1$ ciphertext. Moreover, in both hybrids, the field $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = 1$ dictates the output at time step $t \in \{T', T-2\}$ to be a ciphertext component for time step $T-1$ as argued in Claim A.1.3. Thus, the cPRF key is only required to compute randomness at points $\neq (T-1\|\mathsf{salt})$ for which the punctured root key suffices. Further, it evaluates to the same value as the normal key on all such points in both the hybrids. As a consequence, the decryption values are exactly the same for all the time steps proving the admissibility of $\mathcal{B}$. Thus if $b = 0$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(1, 3)$, while if $b = 1$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(1, 4)$. Hence the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. □

Claim A.1.5. *If* F *is a secure punctured, delegatable* cPRF *scheme, then hybrids* $\mathcal{H}(1, 4)$ *and* $\mathcal{H}(1, 5)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(1, 4)$ and $\mathcal{H}(1, 5)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the punctured, delegatable cPRF scheme F as follows.

1. $\mathcal{B}$ samples $(\mathsf{1FE_1.PK}, \mathsf{1FE_1.MSK}) \leftarrow \mathsf{1FE_1.Setup}(1^\lambda), (\mathsf{1FE_2.PK}, \mathsf{1FE_2.MSK}) \leftarrow \mathsf{1FE_2.Setup}(1^\lambda)$, $\mathsf{salt} \leftarrow \{0, 1\}^\lambda$ and $\mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$. It sends $\mathsf{PK} = \mathsf{1FE_1.PK}$ to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0, 1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It receives K_0^{T-1} on querying for a punctured key at the point $(T - 1\|\mathsf{salt})$ to the cPRF challenger for F.
   (b) $\mathcal{B}$ executes the oblivious TM M on $\mathbf{w}_0$ to learn the (symbol, state) pairs $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$ and $(\sigma_{T-2}^0, \mathsf{q}_{T-2}^0)$ at time steps $T - 1$ and $T - 2$ respectively. It also records the time steps $(T', T - 2)$ and $(T'', T - 3)$ when the individual components of these two (symbol, state) pairs are generated. It then simulates the encryption oracle by computing $\mathsf{CT}_i = \mathsf{1FE_1.Enc}(\mathsf{1FE_1.PK}, x_{1,i})$, where $\forall i \in [\ell], x_{1,i} = (\mathsf{K}_0^{T-1}, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.2. It returns the ciphertext $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[\ell]}$ to $\mathcal{A}$.
   (c) To simulate a function key for M, $\mathcal{B}$ does the following.
      i. It computes $\mathsf{SK}_{\mathsf{ReRand}} \leftarrow \mathsf{1FE_1.KeyGen}(\mathsf{1FE_1.MSK}, \mathsf{ReRand}_{\mathsf{1FE_2.PK},\mathsf{salt},\mathsf{q_{st}},\perp,\perp})$.
      ii. It then computes $\mathsf{1FE_2}$ encodings of $(\sigma_{T-1}^0, \mathsf{q}_{T-1}^0)$, as follows.
         - Compute a delegated key from the punctured root key as $\mathsf{K}_T^{T-1} = \mathsf{F.KeyDel}(\mathsf{K}_0^{T-1}, f_T)$.
         - Query the cPRF challenger at point $(T - 1\|\mathsf{salt})$ to receive an encryption randomness R_E for time step $T - 1$.
         - Compute the $\mathsf{1FE_2}$ *symbol* ciphertext to be given as output at time step T' for the future time step $T-1$ as $\mathsf{CT}_{\mathsf{sym},T-1}^0 = \mathsf{1FE_2.Enc}(\mathsf{1FE_2.PK_1}, \mathbf{z}_1^0; R_E)$, where $\mathbf{z}_1^0 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_T^{T-1}, T - 1, \ell, \sigma_{T-1}^0, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.2.

- Compute the $\mathsf{1FE}_2$ *state* ciphertext to be given as output at time step $T-2$ for the future time step $T-1$ as $\mathsf{CT}^0_{\mathsf{st},T-1} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_2, \mathbf{z}^0_2; R_E)$, where $\mathbf{z}^0_2 = (\mathsf{ST}, \mathsf{q}^0_{T-1})$.

iii. Once it has generated the two $\mathsf{1FE}_2$ ciphertexts $\mathsf{CT}^0_{\mathsf{sym},T-1}$ and $\mathsf{CT}^0_{\mathsf{st},T-1}$, it computes two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{sym},T-1})$ and $\mathsf{ct}_2 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{st},T-1})$.

iv. Finally, it computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that when R_E is computed using K_0 as a pseudorandom value, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(1,4)$, and when R_E is sampled uniformly at random, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(1,5)$. Hence the advantage of $\mathcal{A}$ in distinguishing $\mathcal{H}(1,4)$ and $\mathcal{H}(1,5)$ translates to the advantage of $\mathcal{B}$ in breaking the security of the punctured, delegatable cPRF F. □

Claim A.1.6. *If* $\mathsf{1FE}_2$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(1,5)$ *and* $\mathcal{H}(1,6)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(1,5)$ and $\mathcal{H}(1,6)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_2$ scheme as follows.

1. $\mathcal{B}$ samples $(\mathsf{1FE}_1.\mathsf{PK}, \mathsf{1FE}_1.\mathsf{MSK}) \leftarrow \mathsf{1FE}_1.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and $\mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and gets $\mathsf{1FE}_2.\mathsf{PK}$ from the $\mathsf{1FE}_2$ challenger. It sends $\mathsf{PK} = \mathsf{1FE}_1.\mathsf{PK}$ to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}^\ell_b$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
   (b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the two (symbol, state) pairs $(\sigma^0_{T-1}, \mathsf{q}^0_{T-1})$ and $(\sigma^1_{T-1}, \mathsf{q}^1_{T-1})$ respectively at time step $T-1$. Additionally, $\mathcal{B}$ also learns the (symbol, state) pair $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ that is generated at time step $T-2$ when M is executed on $\mathbf{w}_0$. Further, it records the time steps $(T', T-2)$ and $(T'', T-3)$ when the individual components of these (symbol, state) pairs for $\mathbf{w}_0$ and $\mathbf{w}_1$ are generated and then computes a root key punctured at point $(T-1\|\mathsf{salt})$ as $\mathsf{K}^{T-1}_0 = \mathsf{F.Constrain}(\mathsf{K}_0, (T-1\|\mathsf{salt}))$. It then simulates the encryption oracle by computing $\mathsf{CT}_i = \mathsf{1FE}_1.\mathsf{Enc}(\mathsf{1FE}_1.\mathsf{PK}, x_{1,i})$, where $\forall i \in [\ell], x_{1,i} = (\mathsf{K}^{T-1}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.2. It returns the ciphertext $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[\ell]}$ to $\mathcal{A}$.

(c) To simulate a function key for M, $\mathcal{B}$ does the following.

i. It computes $\mathsf{SK}_{\mathsf{ReRand}} \leftarrow \mathsf{1FE}_1.\mathsf{KeyGen}(\mathsf{1FE}_1.\mathsf{MSK}, \mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp})$.

ii. In order to construct a function key for Next, $\mathcal{B}$ needs to hardwire two SKE ciphertexts which it computes with the help of $\mathsf{1FE}_2$ challenger as follows.

- Delegate the punctured root key to compute $\mathsf{K}_T^{T-1} = \mathsf{F}.\mathsf{KeyDel}(\mathsf{K}_0^{T-1}, f_T)$.
- Create a $\mathsf{1FE}_2$ challenge message pair as $((\mathbf{z}_1^0, \mathbf{z}_2^0), (\mathbf{z}_1^1, \mathbf{z}_2^1))$ such that $\forall b \in \{0,1\}$, $\mathbf{z}_1^b = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_T^{T-1}, T-1, \ell, \sigma_{T-1}^b, \mathsf{Trap}^1)$ and $\mathbf{z}_2^b = (\mathsf{ST}, \mathsf{q}_{T-1}^b)$.
- It sends the challenge message pair $((\mathbf{z}_1^0, \mathbf{z}_2^0), (\mathbf{z}_1^1, \mathbf{z}_2^1))$ to the $\mathsf{1FE}_2$ challenger and gets back $(\mathsf{CT}_{\mathsf{sym},T-1}, \mathsf{CT}_{\mathsf{st},T-1})$.
- It then computes the two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE}.\mathsf{Enc}(\mathsf{K}, \mathsf{CT}_{\mathsf{sym},T-1})$ and $\mathsf{ct}_2 = \mathsf{SKE}.\mathsf{Enc}(\mathsf{K}, \mathsf{CT}_{\mathsf{st},T-1})$.

iii. $\mathcal{B}$ receives $\mathsf{SK}_{\mathsf{Next}}$ for requesting a function key for $\mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{M},\mathsf{ct}_1,\mathsf{ct}_2}$ to the $\mathsf{1FE}_2$ challenger and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the function key queried by $\mathcal{B}$ to the $\mathsf{1FE}_2$ challenger is for a function Next that outputs $\mathsf{1FE}_2$ ciphertexts that are indistinguishable by the security of $\mathsf{1FE}_2$ itself. Therefore, $\mathcal{B}$ is an admissible $\mathsf{1FE}_2$ adversary. Further, when the ciphertext for time step $T-1$ is computed as a $\mathsf{1FE}_2$ encryption of a (symbol, state) pair corresponding to bit $b = 0$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(1,5)$, and when the ciphertext for time step $T-1$ is computed as a $\mathsf{1FE}_2$ encryption of a (symbol, state) pair corresponding to bit $b = 1$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(1,6)$. Thus, the advantage of $\mathcal{A}$ in distinguishing $\mathcal{H}(1,5)$ and $\mathcal{H}(1,6)$ translates to the advantage of $\mathcal{B}$ in breaking the $\mathsf{1FE}_2$ scheme. $\square$

Claim A.1.7. *If* F *is a secure punctured, delegatable* cPRF *scheme, then hybrids* $\mathcal{H}(1,6)$ *and* $\mathcal{H}(1,7)$ *are indistinguishable.*

Proof. The proof is almost identical to Claim A.1.5 where the reduction plays as an adversary against the cPRF challenger and simulates the TMFE adversary $\mathcal{A}$. The only major exceptions now are that $\mathcal{B}$ runs M on both the challenge messages $\mathbf{w}_0$ and $\mathbf{w}_1$ to know the (symbol, state) pairs for both of them at the required time steps for constructing the data structure Trap and that the $\mathsf{1FE}_2$ ciphertext for the symbol and state corresponds to bit $b = 1$. Hence, we omit the details. $\square$

Claim A.1.8. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(1,7)$ *and* $\mathcal{H}(1,8)$ *are indistinguishable.*

Proof. The proof is almost identical to Claim A.1.4 where the reduction plays as an adversary against the $\mathsf{1FE}_1$ challenger and simulates the TMFE adversary $\mathcal{A}$. The only major exceptions now are that $\mathcal{B}$ runs M on both the challenge messages $\mathbf{w}_0$ and $\mathbf{w}_1$ to know the (symbol, state) pairs for both of them at the required time steps for constructing the data structure Trap and that the $\mathsf{1FE}_2$ ciphertext for the symbol and state corresponds to bit $b = 1$. Hence, we omit the details. □

Claim A.1.9. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(1,8)$ *and* $\mathcal{H}(2,1)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(1,8)$ and $\mathcal{H}(2,1)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_1$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE}_1.\mathsf{PK}$ from the $\mathsf{1FE}_1$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and two random strings $\mathsf{ct}_1, \mathsf{ct}_2 \leftarrow \mathcal{C}^{\mathsf{SKE}}$, where $\mathcal{C}^{\mathsf{SKE}}$ denotes the ciphertext space of the SKE scheme.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
   (b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the two (symbol, state) pairs $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ and $(\sigma^1_{T-1}, \mathsf{q}^1_{T-1})$ at time steps $T-2$ and $T-1$ respectively. Further, it records the time steps $(T'', T-3)$ and $(T', T-2)$ when the individual components of these (symbol, state) pairs for $\mathbf{w}_0$ and $\mathbf{w}_1$ respectively are generated and then prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}_0^\ell, \widehat{\mathcal{D}}_1^\ell)$ for the $\mathsf{1FE}_1$ challenger as follows.
      i. For $b = 0$, $\widehat{\mathcal{D}}_0^\ell = \{\mathbf{x}_0 = (x_{0,1}, \ldots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^0)$ with Trap^0 being same as Trap^1 from Figure A.2.
      ii. For $b = 1$, $\widehat{\mathcal{D}}_1^\ell = \{\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with the new fields in Trap^1 as shown in Figure A.3.

mode-real : $\perp$	key-id : salt	$\mathsf{val}_0 : w_{0,i}$	$\mathsf{val}_1 : w_{1,i}$	SKE.K : $\perp$	$\perp$
$\mathsf{mode\text{-}trap}_1$: 1	Target $\mathsf{TS}_1 : T-1$	Sym $\mathsf{TS}_1 : T'$	Sym $\mathsf{val}_1 : \sigma^1_{T-1}$	ST $\mathsf{TS}_1 : T-2$	ST $\mathsf{val}_1 : \mathsf{q}^1_{T-1}$
$\mathsf{mode\text{-}trap}_2$: 1	Target $\mathsf{TS}_2 : T-2$	Sym $\mathsf{TS}_2 : T''$	Sym $\mathsf{val}_2 : \sigma^0_{T-2}$	ST $\mathsf{TS}_2 : T-3$	ST $\mathsf{val}_2 : \mathsf{q}^0_{T-2}$
$\mathsf{mode\text{-}trap}_3 : \perp$	Target TS : $\perp$	Sym TS : $\perp$	$\perp$	ST TS : $\perp$	$\perp$

Figure A.3: Trap^1 configuration in $\mathcal{H}(2,1)$

(c) It sends the distribution pair to the $\mathsf{1FE}_1$ challenger and relays the response back to $\mathcal{A}$.

(d) To simulate a function key for M, $\mathcal{B}$ first requests for a function key to the $\mathsf{1FE}_1$ challenger for the function $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$ and receives $\mathsf{SK}_{\mathsf{ReRand}}$. $\mathcal{B}$ computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ by itself and returns to $\mathcal{A}$ a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that for all time steps $t \notin \{T', T-2\}$, the decryption outputs are exactly the same ciphertexts in both the $\mathcal{H}(1,8)$ and $\mathcal{H}(2,1)$. At a time step $t \in \{T', T-2\}$ in $\mathcal{H}(2,1)$, the decryption *mimics* the decryption of $\mathcal{H}(1,8)$ dictated by $(\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_1 = 1 \wedge \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = \perp)$. More specifically, in $\mathcal{H}(1,8)$ the symbol and state ciphertexts corresponding to time step $T-1$ is first computed by decrypting the SKE ciphertext components hardwired in Next and outputting them at time steps T' and $T-2$ respectively. Alternatively, in $\mathcal{H}(2,1)$, $(\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_1 = 1 \wedge \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = \perp)$ dictates that these ciphertext components are computed with the same randomness at exactly the same time steps T' and $T-2$ respectively. Thus $\mathcal{B}$ is an admissible adversary against the $\mathsf{1FE}_1$ challenger since the outputs for the two challenge message sets are exactly the same. If $b=0$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(1,8)$, while if $b=1$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(2,1)$. Thus the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. □

Claim A.1.10. *If* SKE *is a secure symmetric-key encryption scheme, then hybrids* $\mathcal{H}(2,1)$ *and* $\mathcal{H}(2,2)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(2,1)$ and $\mathcal{H}(2,2)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the SKE scheme as follows.

1. $\mathcal{B}$ samples $(\mathsf{1FE}_1.\mathsf{PK}, \mathsf{1FE}_1.\mathsf{MSK}) \leftarrow \mathsf{1FE}_1.\mathsf{Setup}(1^\lambda)$, $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda)$ and $\mathsf{salt} \leftarrow \{0,1\}^\lambda$. It sends $\mathsf{PK} = \mathsf{1FE}_1.\mathsf{PK}$ to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.

(a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F}.\mathsf{Setup}(1^\lambda)$.

(b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the (symbol, state) pairs $(\sigma_{T-2}^0, \mathsf{q}_{T-2}^0)$ and $(\sigma_{T-1}^1, \mathsf{q}_{T-1}^1)$ at time steps $T-2$ and $T-$

1 respectively. It also records the time steps $(T'', T-3)$ and $(T', T-2)$ when the individual components of these two (symbol, state) pairs are generated. It then simulates the encryption oracle by computing $\mathsf{CT}_i = \mathsf{1FE_1.Enc}(\mathsf{1FE_1.PK}, x_{1,i})$, where $\forall i \in [\ell], x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.3. It returns the ciphertext $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[\ell]}$ to $\mathcal{A}$.

(c) To simulate a function key for M, $\mathcal{B}$ does the following.

  i. It first computes $\mathsf{SK_{ReRand}} \leftarrow \mathsf{1FE_1.KeyGen}(\mathsf{1FE_1.MSK}, \mathsf{ReRand}_{\mathsf{1FE_2.PK},\mathsf{salt},\mathsf{q_{st}},\perp,\perp})$.

  ii. It then computes $\mathsf{1FE_2}$ encodings of $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ as follows.

   - Compute a delegated cPRF key $\mathsf{K}_{T-1} = \mathsf{F.KeyDel}(\mathsf{K}_0, f_{T-1})$ and generate the encryption randomness for time step $T-2$ as $r_{T-2} = \mathsf{F.Eval}(\mathsf{K}_0, (T-2\|\mathsf{salt}))$.

   - Compute the $\mathsf{1FE_2}$ *symbol* ciphertext to be given as output at time step T'' for the future time step $T-2$ as $\mathsf{CT}^0_{\mathsf{sym},T-2} = \mathsf{1FE_2.Enc}(\mathsf{1FE_2.PK_1}, \mathbf{z}^0_1; r_{T-2})$, where $\mathbf{z}^0_1 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_{T-1}, T-2, \ell, \sigma^0_{T-2}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.3.

   - Compute the $\mathsf{1FE_2}$ *state* ciphertext to be given as output at time step $T-3$ for the future time step $T-2$ as $\mathsf{CT}^0_{\mathsf{st},T-2} = \mathsf{1FE_2.Enc}(\mathsf{1FE_2.PK_2}, \mathbf{z}^0_2; r_{T-2})$, where $\mathbf{z}^0_2 = (\mathsf{ST}, \mathsf{q}^0_{T-2})$.

  iii. It sends the $\mathsf{1FE_2}$ ciphertexts $\mathsf{CT}^0_{\mathsf{sym},T-2}, \mathsf{CT}^0_{\mathsf{st},T-2}$ to the challenger of the SKE scheme and gets back $\mathsf{ct}_1, \mathsf{ct}_2$.

  iv. $\mathcal{B}$ then computes $\mathsf{SK_{Next}} \leftarrow \mathsf{1FE_2.KeyGen}(\mathsf{1FE_2.MSK}, \mathsf{Next}_{\mathsf{1FE_2.PK},\mathsf{salt},M,\mathsf{ct_1},\mathsf{ct_2}})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK_{ReRand}}, \mathsf{SK_{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the only difference between the two hybrids is that the SKE ciphertexts hardwired in the function key are random strings in $\mathcal{H}(2,1)$ and are valid SKE encryptions of $(\mathsf{CT}^0_{\mathsf{sym},T-2}, \mathsf{CT}^0_{\mathsf{st},T-2})$ encoding the (symbol, state) pair for time step $T-2$ in $\mathcal{H}(2,2)$. Hence the advantage of an adversary who distinguishes between the two hybrids translates to an advantage of an adversary against the SKE scheme. □

Claim A.1.11. *If* $\mathsf{1FE_1}$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(2,2)$ *and* $\mathcal{H}(2,3)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(2,2)$ and $\mathcal{H}(2,3)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE_1}$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE_1.PK}$ from the $\mathsf{1FE_1}$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE_2.PK}, \mathsf{1FE_2.MSK}) \leftarrow \mathsf{1FE_2.Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda, \mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.

   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.

   (b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the two (symbol, state) pairs $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ and $(\sigma^1_{T-1}, \mathsf{q}^1_{T-1})$ at time steps $T-2$ and $T-1$ respectively. Further, it records the time steps $(T'', T-3)$ and $(T', T-2)$ when the individual components of these (symbol, state) pairs for $\mathbf{w}_0$ and $\mathbf{w}_1$ respectively are generated and then prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}_0^\ell, \widehat{\mathcal{D}}_1^\ell)$ for the $\mathsf{1FE}_1$ challenger as follows.

      i. For $b = 0$, $\widehat{\mathcal{D}}_0^\ell = \{\mathbf{x}_0 = (x_{0,1}, \ldots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^0)$ with Trap^0 being same as Trap^1 from Figure A.3.

      ii. For $b = 1$, $\widehat{\mathcal{D}}_1^\ell = \{\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with the new fields in Trap^1 as shown in Figure A.4.

mode-real : $\perp$	key-id : salt	val$_0$: $w_{0,i}$	val$_1$: $w_{1,i}$	SKE.K : K	$\perp$
mode-trap$_1$: 1	Target TS$_1$: $T-1$	Sym TS$_1$: T'	Sym val$_1$: σ^1_{T-1}	ST TS$_1$: $T-2$	ST val$_1$: q^1_{T-1}
mode-trap$_2$: $\perp$	Target TS$_2$: $\perp$	Sym TS$_2$: $\perp$	Sym val$_2$: $\perp$	ST TS$_2$: $\perp$	ST val$_2$: $\perp$
mode-trap$_3$: 1	Target TS : $T-2$	Sym TS : T''	$\perp$	ST TS : $T-3$	$\perp$

Figure A.4: Trap^1 configuration in $\mathcal{H}(2,3)$

   (c) It sends the distribution pair to the $\mathsf{1FE}_1$ challenger and relays the response back to $\mathcal{A}$.

   (d) To simulate a function key for M, $\mathcal{B}$ does the following.

      i. It requests for a function key for $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$ to the $\mathsf{1FE}_1$ challenger and receives $\mathsf{SK}_{\mathsf{ReRand}}$.

      ii. It then computes $\mathsf{1FE}_2$ encodings of $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$, as follows.

      - Compute a delegated cPRF key $\mathsf{K}_{T-1} = \mathsf{F.KeyDel}(\mathsf{K}_0, f_{T-1})$ and generate the encryption randomness for time step $T-2$ as $r_{T-2} = \mathsf{F.Eval}(\mathsf{K}_0, (T-2\|\mathsf{salt}))$.
      - Compute the $\mathsf{1FE}_2$ *symbol* ciphertext to be given as output at time step T'' for the future time step $T-2$ as $\mathsf{CT}^0_{\mathsf{sym},T-2} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_1, \mathbf{z}^0_1; r_{T-2})$, where $\mathbf{z}^0_1 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_{T-1}, T-2, \ell, \sigma^0_{T-2}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.4.
      - Compute the $\mathsf{1FE}_2$ *state* ciphertext to be given as output at time step $T-3$ for future time step $T-2$ as $\mathsf{CT}^0_{\mathsf{st},T-2} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_2, \mathbf{z}^0_2; r_{T-2})$, where $\mathbf{z}^0_2 = (\mathsf{ST}, \mathsf{q}^0_{T-2})$.

iii. Once it has generated the two $\mathsf{1FE}_2$ ciphertexts $\mathsf{CT}^0_{\mathsf{sym},T-2}$ and $\mathsf{CT}^0_{\mathsf{st},T-2}$, it computes two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{sym},T-2})$ and $\mathsf{ct}_2 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{st},T-2})$.

iv. Finally, it computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Observe that for all time steps $t \notin \{T'', T-3\}$, the decryption outputs are exactly the same ciphertexts in both the $\mathcal{H}(2,2)$ and $\mathcal{H}(2,3)$. At time step $t \in \{T'', T-3\}$ in $\mathcal{H}(2,2)$, ($\mathsf{Trap}^0.\mathsf{mode\text{-}trap}_2 = 1 \wedge \mathsf{Trap}^0.\mathsf{mode\text{-}trap}_3 = \perp$) (in Figure A.3) dictates the decryption to output two decomposed components of a single $\mathsf{1FE}_2$ ciphertext, one component encoding $\mathsf{Trap}^0.\mathsf{Sym}\ \mathsf{val}_2 = \sigma^0_{T-2}$ at time step T'' and the other encoding $\mathsf{Trap}^0.\mathsf{ST}\ \mathsf{val}_2 = \mathsf{q}^0_{T-2}$ at time step $T-3$. Alternatively in $\mathcal{H}(2,3)$, ($\mathsf{Trap}^0.\mathsf{mode\text{-}trap}_2 = \perp \wedge \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = 1$) (in Figure A.4) dictates the decryption to firstly use $\mathsf{Trap}^1.\mathsf{SKE.K} = \mathsf{K}$ to decrypt the hardwired ciphertext ct_1 and output $\mathsf{CT}^0_{\mathsf{sym},T-2}$ at time step T'' (respectively, ct_2 and output $\mathsf{CT}^0_{\mathsf{st},T-2}$ at time step $T-3$). In both the hybrids these symbol and state ciphertext pieces are computed for target time step $T-2$. Thus $\mathcal{B}$ is an admissible $\mathsf{1FE}_1$ adversary. If $b = 0$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(2,2)$, while if $b = 1$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(2,3)$. Hence the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. □

Claim A.1.12. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(2,3)$ *and* $\mathcal{H}(2,4)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(2,3)$ and $\mathcal{H}(2,4)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_1$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE}_1.\mathsf{PK}$ from the $\mathsf{1FE}_1$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and a key $\mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$.

2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}^\ell_b$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.

   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}^\ell_0, \mathcal{D}^\ell_1)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.

(b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the two (symbol, state) pairs $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ and $(\sigma^1_{T-1}, \mathsf{q}^1_{T-1})$ at time steps $T-2$ and $T-1$ respectively. Further, it records the time steps $(T'', T-3)$ and $(T', T-2)$ when the individual components of these (symbol, state) pairs for $\mathbf{w}_0$ and $\mathbf{w}_1$ respectively are generated. It then computes a root key punctured at point $(T-2\|\mathsf{salt})$ as $\mathsf{K}_0^{T-2} = \mathsf{F.Constrain}(\mathsf{K}_0, (T-2\|\mathsf{salt}))$ and prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}^\ell_0, \widehat{\mathcal{D}}^\ell_1)$ for the $\mathsf{1FE}_1$ challenger as follows.

   i. For $b = 0$, $\widehat{\mathcal{D}}^\ell_0 = \{\mathbf{x}_0 = (x_{0,1}, \ldots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ with Trap^1 as per Figure A.4.

   ii. For $b = 1$, $\widehat{\mathcal{D}}^\ell_1 = \{\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0^{T-2}, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with Trap^1 as per Figure A.4.

(c) It sends the distribution pair to the $\mathsf{1FE}_1$ challenger and relays the response back to $\mathcal{A}$.

(d) To simulate a function key for M, $\mathcal{B}$ does the following.

   i. It requests for a function key for $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$ to the $\mathsf{1FE}_1$ challenger and receives $\mathsf{SK}_{\mathsf{ReRand}}$.

   ii. It then computes $\mathsf{1FE}_2$ encodings of $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$, as follows.

      - Compute a punctured, delegated key $\mathsf{K}^{T-2}_{T-1} = \mathsf{F.KeyDel}(\mathsf{K}_0^{T-2}, f_{T-1})$ and generate the encryption randomness for time step $T-2$ as $r_{T-2} = \mathsf{F.Eval}(\mathsf{K}_0, (T-2\|\mathsf{salt}))$.
      - Compute the $\mathsf{1FE}_2$ *symbol* ciphertext to be given as output at time step T'' for the future time step $T-2$ as $\mathsf{CT}^0_{\mathsf{sym},T-2} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_1, \mathbf{z}^0_1; r_{T-2})$, where $\mathbf{z}^0_1 = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}^{T-2}_{T-1}, T-2, \ell, \sigma^0_{T-2}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.4.
      - Compute the $\mathsf{1FE}_2$ *state* ciphertext to be given as output at time step $T-3$ for future time step $T-2$ as $\mathsf{CT}^0_{\mathsf{st},T-2} = \mathsf{1FE}_2.\mathsf{Enc}(\mathsf{1FE}_2.\mathsf{PK}_2, \mathbf{z}^0_2; r_{T-2})$, where $\mathbf{z}^0_2 = (\mathsf{ST}, \mathsf{q}^0_{T-2})$.

   iii. Once it has generated the two $\mathsf{1FE}_2$ ciphertexts $\mathsf{CT}^0_{\mathsf{sym},T-2}$ and $\mathsf{CT}^0_{\mathsf{st},T-2}$, it computes two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{sym},T-2})$ and $\mathsf{ct}_2 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}^0_{\mathsf{st},T-2})$.

   iv. Finally, it computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the only difference in $\mathcal{H}(2,3)$ and $\mathcal{H}(2,4)$ is the replacement of the root cPRF key K_0 with a punctured root key K_0^{T-2} at point $(T-2\|\mathsf{salt})$ in time step $T-2$ in the $\mathsf{1FE}_1$ ciphertext. Moreover, in both the hybrids, the field $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = 1$ dictates the output at time step $t \in \{T'', T-3\}$ to be a ciphertext component for time step $T-2$ as argued in Claim A.1.11. Thus, the cPRF key is only required to compute randomness at points $\neq (T-2\|\mathsf{salt})$ for which the punctured root key suffices. Further, it evaluates to the same value as the normal key on all such points in both the hybrids. As a consequence, the decryption values are exactly the same for all the time steps proving the admissibility of $\mathcal{B}$. Thus if $b = 0$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(2,3)$, while if $b = 1$, $\mathcal{A}$ sees the distribution of $\mathcal{H}(2,4)$. Hence the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. $\square$

Claim A.1.13. *If* F *is a secure punctured, delegatable* cPRF *scheme, then hybrids* $\mathcal{H}(2,4)$ *and* $\mathcal{H}(2,5)$ *are indistinguishable.*

Proof. The proof is almost identical to Claim A.1.5 where the reduction plays as an adversary against the cPRF challenger and simulates the TMFE adversary $\mathcal{A}$ with the following major exceptions.

1. $\mathcal{B}$ runs M on both the sampled messages $\mathbf{w}_0$ and $\mathbf{w}_1$ to know the (symbol, state) pairs at the time steps $T-2$ and $T-1$ respectively for constructing the data structure Trap as in $\mathcal{H}(2,4)$. The challenge ciphertext encodes K_0^{T-2}, i.e., a root key punctured at point $(T-2\|\mathsf{salt})$.
2. The cPRF challenger is queried at the point $(T-2\|\mathsf{salt})$ to receive an encryption randomness for time step $T-2$. This is used in computing the $\mathsf{1FE}_2$ ciphertext encoding the (symbol, state) pair generated at time steps $(T'', T-3)$ for time step $T-2$ when M is run on $\mathbf{w}_0$.

The other details follow as before and hence we omit them. $\square$

Claim A.1.14. *If* $\mathsf{1FE}_2$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(2,5)$ *and* $\mathcal{H}(2,6)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(2,5)$ and $\mathcal{H}(2,6)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_2$ scheme as follows.

1. $\mathcal{B}$ samples $(\mathsf{1FE}_1.\mathsf{PK}, \mathsf{1FE}_1.\mathsf{MSK}) \leftarrow \mathsf{1FE}_1.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and $\mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and gets $\mathsf{1FE}_2.\mathsf{PK}$ from the $\mathsf{1FE}_2$ challenger. It sends $\mathsf{PK} = \mathsf{1FE}_1.\mathsf{PK}$ to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.

(b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the two (symbol, state) pairs $(\sigma^0_{T-2}, \mathsf{q}^0_{T-2})$ and $(\sigma^1_{T-2}, \mathsf{q}^1_{T-2})$ respectively at time step $T-2$. Additionally, $\mathcal{B}$ also learns the (symbol, state) pair $(\sigma^1_{T-1}, \mathsf{q}^1_{T-1})$ that is generated at time step $T-1$ when M is executed on $\mathbf{w}_1$. Further, it records the time steps $(T'', T-3)$ and $(T', T-2)$ when the individual components of these (symbol, state) pairs for $\mathbf{w}_0$ and $\mathbf{w}_1$ are generated and then computes a root key punctured at point $(T-2\|\mathsf{salt})$ as $\mathsf{K}_0^{T-2} = \mathsf{F}.\mathsf{Constrain}(\mathsf{K}_0, (T-2\|\mathsf{salt}))$. It then simulates the encryption oracle by computing $\mathsf{CT}_i = \mathsf{1FE}_1.\mathsf{Enc}(\mathsf{1FE}_1.\mathsf{PK}, x_{1,i})$, where $\forall i \in [\ell], x_{1,i} = (\mathsf{K}_0^{T-2}, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ and Trap^1 is as per Figure A.4. It returns the ciphertext $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[\ell]}$ to $\mathcal{A}$.

(c) To simulate a function key for M, $\mathcal{B}$ does the following.

i. It computes $\mathsf{SK}_{\mathsf{ReRand}} \leftarrow \mathsf{1FE}_1.\mathsf{KeyGen}(\mathsf{1FE}_1.\mathsf{MSK}, \mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp})$.

ii. In order to construct a function key for Next, $\mathcal{B}$ needs to hardwire two SKE ciphertexts which it computes with the help of $\mathsf{1FE}_2$ challenger as follows.

- Delegate the punctured root key to compute $\mathsf{K}_{T-1}^{T-2} = \mathsf{F}.\mathsf{KeyDel}(\mathsf{K}_0^{T-2}, f_{T-1})$.
- Create a $\mathsf{1FE}_2$ challenge message pair as $((\mathbf{z}_1^0, \mathbf{z}_2^0), (\mathbf{z}_1^1, \mathbf{z}_2^1))$ such that $\forall b \in \{0,1\}$, $\mathbf{z}_1^b = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_{T-1}^{T-2}, T-2, \ell, \sigma^b_{T-2}, \mathsf{Trap}^1)$ and $\mathbf{z}_2^b = (\mathsf{ST}, \mathsf{q}^b_{T-2})$, where Trap^1 is as per Figure A.4.
- It sends the challenge message pair $((\mathbf{z}_1^0, \mathbf{z}_2^0), (\mathbf{z}_1^1, \mathbf{z}_2^1))$ to the $\mathsf{1FE}_2$ challenger and gets back $(\mathsf{CT}_{\mathsf{sym},T-2}, \mathsf{CT}_{\mathsf{st},T-2})$.
- It then computes the two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE}.\mathsf{Enc}(\mathsf{K}, \mathsf{CT}_{\mathsf{sym},T-2})$ and $\mathsf{ct}_2 = \mathsf{SKE}.\mathsf{Enc}(\mathsf{K}, \mathsf{CT}_{\mathsf{st},T-2})$.

iii. $\mathcal{B}$ receives $\mathsf{SK}_{\mathsf{Next}}$ for requesting a function key for $\mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{M},\mathsf{ct}_1,\mathsf{ct}_2}$ to the $\mathsf{1FE}_2$ challenger and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the function key queried by $\mathcal{B}$ to the $\mathsf{1FE}_2$ challenger is for a function Next that outputs $\mathsf{1FE}_2$ ciphertexts that are indistinguishable by the security of $\mathsf{1FE}_2$ itself. Therefore, $\mathcal{B}$ is an admissible $\mathsf{1FE}_2$ adversary. Further, when the ciphertext for time step $T-2$ is computed as a $\mathsf{1FE}_2$ encryption of a (symbol, state) pair corresponding to bit $b=0$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(2,5)$, and when the ciphertext for time step $T-2$ is computed as a $\mathsf{1FE}_2$ encryption of a (symbol, state) pair corresponding to bit $b=1$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(2,6)$. Thus, the advantage of $\mathcal{A}$ in distinguishing $\mathcal{H}(2,5)$ and $\mathcal{H}(2,6)$ translates to the advantage of $\mathcal{B}$ in breaking the $\mathsf{1FE}_2$ scheme. □

Claim A.1.15. *If* F *is a secure punctured, delegatable* cPRF *scheme, then hybrids* $\mathcal{H}(2,6)$ *and* $\mathcal{H}(2,7)$ *are indistinguishable.*

Proof. The proof is similar to Claim A.1.7 and hence we omit the details. □

Claim A.1.16. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(2,7)$ *and* $\mathcal{H}(2,8)$ *are indistinguishable.*

Proof. The proof is similar to Claim A.1.8 and hence we omit the details. □

Claim A.1.17. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(2,8)$ *and* $\mathcal{H}(3,1)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(2,8)$ and $\mathcal{H}(3,1)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_1$ scheme as follows.

1. $\mathcal{B}$ receives $\mathsf{1FE}_1.\mathsf{PK}$ from the $\mathsf{1FE}_1$ challenger and returns this to $\mathcal{A}$. Additionally, it samples by itself $(\mathsf{1FE}_2.\mathsf{PK}, \mathsf{1FE}_2.\mathsf{MSK}) \leftarrow \mathsf{1FE}_2.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and two random strings $\mathsf{ct}_1, \mathsf{ct}_2 \leftarrow \mathcal{C}^{\mathsf{SKE}}$, where $\mathcal{C}^{\mathsf{SKE}}$ denotes the ciphertext space of the SKE scheme.
2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M obeying the admissibility criteria that $\forall b \in \{0,1\}, \mathbf{w}_b \leftarrow \mathcal{D}_b^\ell$, $\mathsf{runtime}(M, \mathbf{w}_0) = \mathsf{runtime}(M, \mathbf{w}_1)$ and $M(\mathbf{w}_0) \overset{c}{\approx} M(\mathbf{w}_1)$, $\mathcal{B}$ does the following.
    (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
    (b) $\mathcal{B}$ executes the oblivious TM M on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$ to learn the two (symbol, state) pairs $(\sigma^0_{T-3}, \mathsf{q}^0_{T-3})$ and $(\sigma^1_{T-2}, \mathsf{q}^1_{T-2})$ at time steps $T-3$ and $T-2$ respectively. Further, it records the time steps $(T''', T-4)$ and $(T'', T-3)$ when the individual components of these (symbol, state) pairs for $\mathbf{w}_0$ and $\mathbf{w}_1$ respectively are generated and then prepares a new pair of challenge distributions $(\widehat{\mathcal{D}}_0^\ell, \widehat{\mathcal{D}}_1^\ell)$ for the $\mathsf{1FE}_1$ challenger as follows.
        i. For $b = 0$, $\widehat{\mathcal{D}}_0^\ell = \{\mathbf{x}_0 = (x_{0,1}, \ldots, x_{0,\ell})\}$, where $\forall i \in [\ell]$ $x_{0,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^0)$ with Trap^0 being same as Trap^1 from Figure A.4.
        ii. For $b = 1$, $\widehat{\mathcal{D}}_1^\ell = \{\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,\ell})\}$, where $\forall i \in [\ell]$ $x_{1,i} = (\mathsf{K}_0, i, \ell, w_{0,i}, \mathsf{Trap}^1)$, with the new fields in Trap^1 as shown in Figure A.5.

mode-real : $\perp$	key-id : salt	val_0 : $w_{0,i}$	val_1 : $w_{1,i}$	SKE.K : $\perp$	$\perp$
$\mathsf{mode\text{-}trap}_1$: 1	Target TS_1 : $T-2$	Sym TS_1 : T''	Sym val_1 : σ^1_{T-2}	ST TS_1 : $T-3$	ST val_1 : q^1_{T-2}
$\mathsf{mode\text{-}trap}_2$: 1	Target TS_2 : $T-3$	Sym TS_2 : T'''	Sym val_2 : σ^0_{T-3}	ST TS_2 : $T-4$	ST val_2 : q^0_{T-3}
$\mathsf{mode\text{-}trap}_3$: $\perp$	Target TS : $\perp$	Sym TS : $\perp$	$\perp$	ST TS : $\perp$	$\perp$

Figure A.5: Trap^1 configuration in $\mathcal{H}(3,1)$

(c) It sends the distribution pair to the $\mathsf{1FE}_1$ challenger and relays the response back to $\mathcal{A}$.

(d) To simulate a function key for M, $\mathcal{B}$ first requests for a function key to the $\mathsf{1FE}_1$ challenger for the function $\mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\perp,\perp}$ and receives $\mathsf{SK}_{\mathsf{ReRand}}$. $\mathcal{B}$ computes $\mathsf{SK}_{\mathsf{Next}} \leftarrow \mathsf{1FE}_2.\mathsf{KeyGen}(\mathsf{1FE}_2.\mathsf{MSK}, \mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},M,\mathsf{ct}_1,\mathsf{ct}_2})$ by itself and returns to $\mathcal{A}$ a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$.

Note that the (symbol, state) pair for time step $T-2$ has already been switched to correspond to $b=1$ from the prior hybrid. Thus, maintaining the trapdoor information for time step $T-1$ is now redundant and follows by normal decryption from time step $T-2$. The (symbol,state) pair for time step $T-3$ now corresponds to bit $b=0$ and therefore the decryption chain inconsistency arises at time step $T-2$ now. Hence, intuitively we “slide" the trapdoor by replacing a new trapdoor data structure in $\mathcal{H}(3,1)$ in a way that still maintains functional equivalence with $\mathcal{H}(2,8)$ at all the time steps but contains hardwired information about the time steps $T-3$ and $T-2$ now. We show the admissibility of the reduction $\mathcal{B}$ as follows.

Observe that for all time steps $t \notin \{T'', T-3, T''', T-4\}$, the decryption outputs are exactly the same sequence of ciphertexts in both the hybrids which are output by the normal decryption. At a time step $t \in \{T'', T-3, T''', T-4\}$ in $\mathcal{H}(3,1)$, the decryption is dictated by $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_1 = \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_2 = 1$. In particular, the ciphertext components for time step $T-2$ corresponding to $b=1$ is output at time steps T'' and $T-3$ and is triggered by $\mathsf{Trap}^0.\mathsf{mode\text{-}trap}_3 = 1$ in $\mathcal{H}(2,8)$ and $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_1 = 1$ in $\mathcal{H}(3,1)$. On the other hand, the ciphertext components for time step $T-3$ corresponding to $b=0$ is output at time steps T''' and $T-4$ and is triggered by the normal decryption in $\mathcal{H}(2,8)$ and by $\mathsf{Trap}^0.\mathsf{mode\text{-}trap}_2 = 1$ in $\mathcal{H}(3,1)$. The ciphertext components for time step $T-1$ corresponding to $b=1$ is output at time steps T' and $T-2$ and is triggered by $\mathsf{Trap}^0.\mathsf{mode\text{-}trap}_1 = 1$ in $\mathcal{H}(2,8)$ and by the normal decryption (as a consequence of already having the outputs at time step $T-2$ switched to $b=1$) in $\mathcal{H}(3,1)$. Further, note that all these ciphertext components are exactly the same for both the hybrids.

Therefore, $\mathcal{B}$ is an admissible adversary against the $\mathsf{1FE}_1$ challenger since the outputs for the two challenge message sets are exactly the same. Hence $\mathcal{A}$ sees the distribution of $\mathcal{H}(2,8)$, if $b=0$, and that of $\mathcal{H}(3,1)$, if $b=1$. Thus the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. $\square$

Note that $\mathcal{H}(3,i)$ is analogous to $\mathcal{H}(2,i), \forall i \in [8]$. Now consider any pair of challenge message vectors $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1},\ldots,w_{b,\ell})\}_{b\in\{0,1\}}$ of arbitrary length ℓ with any TM M taking T time steps to halt on either inputs. In general, we have that $\mathcal{H}(t,i)$ is analogous to $\mathcal{H}(t-1,i)$, for all $t \in [3, T-(\ell+1)], i \in [8]$. Observe further that for any given $k \in [3, T-(\ell+1)]$, we have the following computational indistinguishability chain via the intermediate hybrids.

$$\mathcal{H}(k-1,1) \overset{c}{\approx} \mathcal{H}(k-1,2) \overset{c}{\approx} \cdots \overset{c}{\approx} \mathcal{H}(k-1,8) \overset{c}{\approx} \mathcal{H}(k,1) \overset{c}{\approx} \mathcal{H}(k,2) \overset{c}{\approx} \cdots \overset{c}{\approx} \mathcal{H}(k,8)$$

We can easily extend this computational indistinguishability chain further to have the following.

$$\mathcal{H}(0) \overset{c}{\approx} \mathcal{H}(1,1) \overset{c}{\approx} \mathcal{H}(1,8) \overset{c}{\approx} \mathcal{H}(2,1) \overset{c}{\approx} \mathcal{H}(2,8) \overset{c}{\approx} \cdots \overset{c}{\approx} \mathcal{H}(T-(\ell+1),1) \overset{c}{\approx} \mathcal{H}(T-(\ell+1),8)$$

Note that in $\mathcal{H}(T-(\ell+1),8)$, the (symbol, state) pair corresponding to the output at time step $\ell+1$ has already been switched to $b=1$. Proceeding one step backward in the execution chain we reach time step ℓ where the $\mathsf{1FE}_2$ ciphertext components are computed partially by each of $\mathsf{SK}_{\mathsf{ReRand}}$ and $\mathsf{SK}_{\mathsf{Next}}$. More specifically, at any time step $j \in [2,\ell]$ the $\mathsf{1FE}_2$ ciphertext component encoding the “symbol" w_j is output by ReRand. Accordingly, the $\mathsf{1FE}_2$ ciphertext component encoding the “state" q_j for the same time step j is output by Next only when it gets (w_{j-1}, q_{j-1}) as input, i.e., the symbol and state at time step $j-1$, each of which is encrypted with the exact same randomness. Hence, to proceed with the security proof at any time step $j \in [2,\ell]$, while switching from $b=0$ to $b=1$ the reduction $\mathcal{B}$ simulating $\mathsf{1FE}_1$ itself will now hardwire the SKE ciphertext encoding $\mathsf{1FE}_2.\mathsf{CT}(w_{b,j})$ into ReRand and the SKE ciphertext encoding $\mathsf{1FE}_2.\mathsf{CT}(\mathsf{q}_j^b)$ into Next after receiving them from the $\mathsf{1FE}_2$ challenger. At time step $j=1$, $\mathcal{B}$ hardwires the SKE ciphertext encoding both $\mathsf{1FE}_2.\mathsf{CT}(w_{b,1})$ and $\mathsf{1FE}_2.\mathsf{CT}(\mathsf{q}_{\mathsf{st}})$ into ReRand function only. This is since ReRand outputs the (symbol, state) ciphertext pair at the first time step as per functionality. Indistinguishability between these hybrids is as before.

Similar to the transition from $\mathcal{H}(2,8)$ to $\mathcal{H}(3,1)$, at time step $j=\ell+1$ we slide the trapdoor to switch the ciphertext in slot 1 for time step $\ell+1$ (corresponding to $b=1$) and slot 2 for time step ℓ (corresponding to $b=0$). We also set $\mathsf{Trap}^1.\mathsf{mode\text{-}trap}_3 = \perp, \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_1 = \mathsf{Trap}^1.\mathsf{mode\text{-}trap}_2 = 1$. The decryption values being exactly the same in $\mathcal{H}(T-(\ell+1),8)$ and $\mathcal{H}(T-\ell,1)$, security follows from $\mathsf{1FE}_1$.

However, once we reach time step ℓ at $\mathcal{H}(T-\ell,8)$ when the bit b (for time step ℓ) has already been switched from 0 to 1 and we are about to slide the trapdoor to go to the next hybrid, we must add an additional hybrid $\mathcal{H}(T-j,9)$ for all $j \in [1,\ell]$, as discussed in Section 2.7.3, namely:

$\mathcal{H}(T-j,9)$: In this hybrid, we modify the $\mathsf{1FE}_1$ challenge ciphertext in position j as follows: the encoded message is changed corresponding to $b=1$ and flag $\mathsf{mode\text{-}real}=1$. The other flags $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = \mathsf{mode\text{-}trap}_3 = \perp$.

Consider $j=\ell$. Note that all ciphertexts previous to time step ℓ remain unchanged, and output their corresponding symbol ciphertexts correctly. The Next circuit outputs the state ciphertext for time step ℓ corresponding to bit $b=1$. The only difference between this hybrid and the previous one is that here we use the real mode to output the symbol ciphertext for $b=1$ whereas previously we used the trapdoor mode to output the same symbol CT. Hence, decryption values in both hybrids are exactly the same, and indistinguishability follows from security of $\mathsf{1FE}_1$.

As before from $\mathcal{H}(1,8)$ to $\mathcal{H}(2,5)$ and $\mathcal{H}(2,6)$ to $\mathcal{H}(3,1)$, we get two similar sequence of hybrids from $\mathcal{H}(T-(\ell+1),8)$ to $\mathcal{H}(T-\ell,5)$ and from $\mathcal{H}(T-\ell,6)$ to $\mathcal{H}(T-\ell,8)$. Additionally, we now go from $\mathcal{H}(T-\ell,8)$ to $\mathcal{H}(T-(\ell-1),1)$ via the intermediate extra hybrid $\mathcal{H}(T-\ell,9)$ as follows.

$$\mathcal{H}(T-(\ell+1),8) \overset{\mathsf{1FE}_1}{\overset{c}{\approx}} \mathcal{H}(T-\ell,1) \overset{\mathsf{SKE}}{\overset{c}{\approx}} \mathcal{H}(T-\ell,2) \overset{\mathsf{1FE}_1}{\overset{c}{\approx}} \mathcal{H}(T-\ell,3) \overset{\mathsf{1FE}_1}{\overset{c}{\approx}}$$
$$\mathcal{H}(T-\ell,4) \overset{\mathsf{cPRF}}{\overset{c}{\approx}} \mathcal{H}(T-\ell,5)$$

$$\mathcal{H}(T-\ell,6) \overset{\mathsf{cPRF}}{\overset{c}{\approx}} \mathcal{H}(T-\ell,7) \overset{\mathsf{1FE}_1}{\overset{c}{\approx}} \mathcal{H}(T-\ell,8) \quad \text{and} \quad \underbrace{\mathcal{H}(T-\ell,8) \overset{\mathsf{1FE}_1}{\overset{c}{\approx}} \mathcal{H}(T-\ell,9)} \overset{\mathsf{1FE}_1}{\overset{c}{\approx}}$$
$$\mathcal{H}(T-(\ell-1),1)$$

In the following claims, we show a formal reduction between $\mathcal{H}(T-\ell,5)$ and $\mathcal{H}(T-\ell,6)$ and sketch a high level proof of computational indistinguishability for that of between $\mathcal{H}(T-\ell,8)$ and $\mathcal{H}(T-\ell,9)$ thereby connecting the above computational indistinguishability chains into one when the symbol and state at time step ℓ gets switched from $b=0$ to $b=1$ finally.

Claim A.1.18. *If* $\mathsf{1FE}_2$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(T-\ell,5)$ *and* $\mathcal{H}(T-$

$\ell, 6)$ *are indistinguishable.*

Proof. Given a PPT adversary $\mathcal{A}$ that distinguishes $\mathcal{H}(T-\ell, 5)$ and $\mathcal{H}(T-\ell, 6)$, we construct another PPT adversary $\mathcal{B}$ who breaks the security of the $\mathsf{1FE}_2$ scheme as follows.

1. $\mathcal{B}$ samples $(\mathsf{1FE}_1.\mathsf{PK}, \mathsf{1FE}_1.\mathsf{MSK}) \leftarrow \mathsf{1FE}_1.\mathsf{Setup}(1^\lambda), \mathsf{salt} \leftarrow \{0,1\}^\lambda$ and $\mathsf{K} \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and gets $\mathsf{1FE}_2.\mathsf{PK}$ from the $\mathsf{1FE}_2$ challenger. It sends $\mathsf{PK} = \mathsf{1FE}_1.\mathsf{PK}$ to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs a pair of challenge distributions $(\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ with support Σ^ℓ for any arbitrary $\ell = \mathsf{poly}(\lambda)$ and a function query M which obeys the admissibility criteria, $\mathcal{B}$ does the following.
   (a) To simulate the challenge ciphertext, it first samples a pair of challenge messages $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow (\mathcal{D}_0^\ell, \mathcal{D}_1^\ell)$ such that $\{\mathbf{w}_b\}_{b\in\{0,1\}} = \{(w_{b,1}, \ldots, w_{b,\ell})\}_{b\in\{0,1\}}$. It also samples a root cPRF key $\mathsf{K}_0 \leftarrow \mathsf{F.Setup}(1^\lambda)$.
   (b) $\mathcal{B}$ learns the two (symbol, state) pairs at time step ℓ when the oblivious TM M is run on *both* $\mathbf{w}_0$ and $\mathbf{w}_1$. Denote these pairs as $(\sigma_\ell^0, \mathsf{q}_\ell^0)$ and $(\sigma_\ell^1, \mathsf{q}_\ell^1)$, where $\sigma_\ell^b = w_{b,\ell}$ now. $\mathcal{B}$ also learns the (symbol, state) pair at time step $\ell+1$ when M is run on $\mathbf{w}_1$ and denote this pair as $(\sigma_{\ell+1}^1, \mathsf{q}_{\ell+1}^1)$. Further, it also records the time steps $(\ell, \ell-1)$ and $(\ell', \ell), \ell' \leq \ell$ when the individual components of the (symbol, state) pairs $(\sigma_\ell^b, \mathsf{q}_\ell^b), \forall b \in \{0,1\}$ and $(\sigma_{\ell+1}^1, \mathsf{q}_{\ell+1}^1)$ respectively are generated and then computes a root key punctured at point $(\ell \| \mathsf{salt})$ as $\mathsf{K}_0^\ell = \mathsf{F.Constrain}(\mathsf{K}_0, (\ell\|\mathsf{salt}))$. It then simulates the encryption oracle by computing $\mathsf{CT}_i = \mathsf{1FE}_1.\mathsf{Enc}(\mathsf{1FE}_1.\mathsf{PK}, x_{1,i})$, where $\forall i \in [\ell], x_{1,i} = (\mathsf{K}_0^\ell, i, \ell, w_{0,i}, \mathsf{Trap}^1)$ and Trap^1 is as per $\mathcal{H}(T-\ell, 3)$ shown in Figure A.6. It returns the ciphertext $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[\ell]}$ to $\mathcal{A}$.

$\mathsf{mode\text{-}real} : \bot$	$\mathsf{key\text{-}id} : \mathsf{salt}$	$\mathsf{val}_0 : w_{0,i}$	$\mathsf{val}_1 : w_{1,i}$	$\mathsf{SKE.K} : \mathsf{K}$	$\bot$
$\mathsf{mode\text{-}trap}_1 : 1$	$\mathsf{Target\ TS}_1 : \ell+1$	$\mathsf{Sym\ TS}_1 : \ell'$	$\mathsf{Sym\ val}_1 : \sigma_{\ell+1}^1$	$\mathsf{ST\ TS}_1 : \ell$	$\mathsf{ST\ val}_1 : \mathsf{q}_{\ell+1}^1$
$\mathsf{mode\text{-}trap}_2 : \bot$	$\mathsf{Target\ TS}_2 : \bot$	$\mathsf{Sym\ TS}_2 : \bot$	$\mathsf{Sym\ val}_2 : \bot$	$\mathsf{ST\ TS}_2 : \bot$	$\mathsf{ST\ val}_2 : \bot$
$\mathsf{mode\text{-}trap}_3 : 1$	$\mathsf{Target\ TS} : \ell$	$\mathsf{Sym\ TS} : \ell$	$\bot$	$\mathsf{ST\ TS} : \ell-1$	$\bot$

Figure A.6: Trap^1 configuration in $\mathcal{H}(T-\ell, 3)$

   (c) To simulate a function key for M, $\mathcal{B}$ does the following.
      i. In order to construct a function key for ReRand and Next, $\mathcal{B}$ now needs to hardwire an SKE ciphertext in each of the functions which it computes with the help of $\mathsf{1FE}_2$ challenger as follows.
         - Delegate the punctured root key to compute $\mathsf{K}_{\ell+1}^\ell = \mathsf{F.KeyDel}(\mathsf{K}_0^\ell, f_{\ell+1})$.
         - Create a $\mathsf{1FE}_2$ challenge message pair as $((\mathbf{z}_1^0, \mathbf{z}_2^0), (\mathbf{z}_1^1, \mathbf{z}_2^1))$ such that $\forall b \in \{0,1\}$, $\mathbf{z}_1^b = (\mathsf{SYM}, \mathsf{salt}, \mathsf{K}_{\ell+1}^\ell, \ell, \ell, \sigma_\ell^b, \mathsf{Trap}^1)$ and $\mathbf{z}_2^b = (\mathsf{ST}, \mathsf{q}_\ell^b)$.
         - It sends the challenge message pair $((\mathbf{z}_1^0, \mathbf{z}_2^0), (\mathbf{z}_1^1, \mathbf{z}_2^1))$ to the $\mathsf{1FE}_2$ challenger and gets back $(\mathsf{CT}_{\mathsf{sym},\ell}, \mathsf{CT}_{\mathsf{st},\ell})$.

- It then computes the two SKE ciphertexts $\mathsf{ct}_1 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}_{\mathsf{sym},\ell})$ and $\mathsf{ct}_2 = \mathsf{SKE.Enc}(\mathsf{K}, \mathsf{CT}_{\mathsf{st},\ell})$.

ii. $\mathcal{B}$ computes $\mathsf{SK}_{\mathsf{ReRand}} \leftarrow \mathsf{1FE}_1.\mathsf{KeyGen}(\mathsf{1FE}_1.\mathsf{MSK}, \mathsf{ReRand}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{q}_{\mathsf{st}},\mathsf{ct}_1,\perp})$ by itself.

iii. $\mathcal{B}$ receives $\mathsf{SK}_{\mathsf{Next}}$ for requesting a function key for $\mathsf{Next}_{\mathsf{1FE}_2.\mathsf{PK},\mathsf{salt},\mathsf{M},\perp,\mathsf{ct}_2}$ to the $\mathsf{1FE}_2$ challenger and returns a function key for M as $\mathsf{SK}_M = (\mathsf{SK}_{\mathsf{ReRand}}, \mathsf{SK}_{\mathsf{Next}})$ to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ does the same.

Note that the function key queried by $\mathcal{B}$ to the $\mathsf{1FE}_2$ challenger is for a function Next that outputs $\mathsf{1FE}_2$ ciphertexts that are indistinguishable by the security of $\mathsf{1FE}_2$ itself. Therefore, $\mathcal{B}$ is an admissible $\mathsf{1FE}_2$ adversary. Further, when the ciphertext for time step ℓ is computed as a $\mathsf{1FE}_2$ encryption of a (symbol, state) pair corresponding to bit $b = 0$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(T-\ell, 5)$, and when the ciphertext for time step ℓ is computed as a $\mathsf{1FE}_2$ encryption of a (symbol, state) pair corresponding to bit $b = 1$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(T-\ell, 6)$. Hence the advantage of $\mathcal{A}$ in distinguishing $\mathcal{H}(T-\ell, 5)$ and $\mathcal{H}(T-\ell, 6)$ translates to the advantage of $\mathcal{B}$ in breaking the $\mathsf{1FE}_2$ scheme. $\square$

Claim A.1.19. *If* $\mathsf{1FE}_1$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(T-\ell, 8)$ *and* $\mathcal{H}(T-\ell, 9)$ *are indistinguishable.*

Proof. We describe the proof at a high level and omit the details. Note that all ciphertexts previous to time step ℓ remain unchanged, and output their corresponding symbol ciphertexts correctly. The Next circuit outputs the state ciphertext for time step ℓ corresponding to bit $b = 1$. The only difference between this hybrid and the previous one is that here we use the real mode to output the symbol ciphertext for $b = 1$ whereas previously we used the trapdoor mode to output the same symbol ciphertext. Hence, decryption values in both hybrids are exactly the same. When the $\mathsf{1FE}_1$ ciphertext for time step ℓ is computed corresponding to $b = 0$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(T-\ell, 8)$, and when the $\mathsf{1FE}_1$ ciphertext for time step ℓ is computed corresponding to $b = 1$, $\mathcal{A}$'s view is identical to that of $\mathcal{H}(T-\ell, 9)$. Hence the advantage of $\mathcal{A}$ in distinguishing $\mathcal{H}(T-\ell, 8)$ and $\mathcal{H}(T-\ell, 9)$ translates to the advantage of $\mathcal{B}$ in breaking the $\mathsf{1FE}_1$ scheme. $\square$

Denoting $\tau = (T - j)$ for any $j \in [\ell]$, we get a sequence of hybrids shown below, where we define $\mathcal{H}(T,1) \triangleq \mathcal{H}(T)$ and have the final Claim A.1.20 which completes the proof of Theorem 2.7.1.

$$\mathcal{H}(\tau,8) \overset{\mathsf{1FE_1}}{\overset{c}{\approx}} \mathcal{H}(\tau,9) \overset{\mathsf{1FE_1}}{\overset{c}{\approx}} \mathcal{H}(\tau+1,1) \overset{\mathsf{SKE}}{\overset{c}{\approx}} \cdots \overset{\mathsf{cPRF}}{\overset{c}{\approx}} \mathcal{H}(\tau+1,5) \overset{\mathsf{1FE_2}}{\overset{c}{\approx}} \mathcal{H}(\tau+1,6) \overset{\mathsf{cPRF}}{\overset{c}{\approx}} \cdots \overset{\mathsf{1FE_1}}{\overset{c}{\approx}}$$
$$\mathcal{H}(\tau+1,8) \overset{\mathsf{1FE_1}}{\overset{c}{\approx}} \mathcal{H}(\tau+1,9) \overset{\mathsf{1FE_1}}{\overset{c}{\approx}} \mathcal{H}(\tau+2,1)$$

Claim A.1.20. *If* $\mathsf{1FE_1}$ *is a secure* CktFE *scheme, then hybrids* $\mathcal{H}(T-1,9)$ *and* $\mathcal{H}(T)$ *are indistinguishable.*

Proof. Note that $\mathsf{Trap}^1.\mathsf{mode\text{-}real} = 1$ for ciphertexts in both worlds. The only difference between both these hybrids is that in the former Trap contains other information whereas in the latter all other fields disabled with $\perp$. However, since $\mathsf{Trap}^1.\mathsf{mode\text{-}real} = 1$, these fields anyway play no role in decryption, so the decryption values stay the same.

Selective Security. The above proof shows security as per the weak selective definition, in which the adversary submits the challenge messages and keys at the same time. This can be easily strengthened to selective security in which the key requests can be made after seeing the challenge ciphertext. Since the full selective game requires an additional trapdoor structure, we did not show it here for ease of exposition, as the current proof is already quite complex. Note that currently, the proof is restricted to weak selective because in order to program the symbol and state messages for some time step in the Trap data structure, the machine which produces these symbol, state pairs must be specified. This dependency may be easily overcome by instead having an additional trapdoor data structure in the key, which contains the above information. Thus, the challenge ciphertext can be programmed without knowledge of the keys, and selective security can be achieved. □

A.2 Missing Details in Proof of Theorem 2.8.1

The modified trapdoor data structure is shown in Figure A.7. There is an additional field that records the global salt value.

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

Figure A.7: Data Structure Trap used for Proof

The Hybrids. We consider the case where the adversary makes a single key query but makes Q ciphertext queries in each co-ordinate. We assume a lexicographic ordering over the Q^k global salt values, and denote by gsalt_j the j^{th} member of this sequence.

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

For $j \in [Q^k]$, do:

$\mathcal{H}(j, 1, 1)$: In this world, all ciphertexts (constructed by the encryptor as well as function keys) have $\mathsf{mode\text{-}real} = \perp$, $\mathsf{mode\text{-}trap}_1 = 1,\ \mathsf{mode\text{-}trap}_2 = 1,\ \mathsf{mode\text{-}trap}_3 = \perp$. We program the last link in the decryption chain corresponding to gsalt_j for switching bit b by setting:

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

The fields Sym TS_1 and ST TS_1 contain the time steps when the symbol and state ciphertext pieces are generated for time step $T - 1$, and the fields Sym val_1 and ST val_1 contain the symbol and state values which must be encrypted by the function key in the above time steps when $\mathsf{mode\text{-}trap}_1$ is set.

Indistinguishability follows from security of kFE, since the decryption values in both hybrids are exactly the same.

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

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

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

$T-1$ ciphertext corresponding to execution thread gsalt_j.

Indistinguishability follows from security of kFE, since the decryption values in both hybrids are exactly the same.

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

Indistinguishability follows from security of kFE.

$\mathcal{H}(j,1,5)$: Switch the randomness in the 1FE ciphertexts which are hardwired in the key to true randomness.

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

$\mathcal{H}(j,1,6)$: Switch the value encoded in the 1FE ciphertexts which are hardwired in the key to correspond to $b=1$.

Indistinguishability follows from security of 1FE.

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

Indistinguishability follows from security of cPRF as discussed above.

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

Indistinguishability follows from security of kFE as discussed above.

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

Indistinguishability follows from security of kFE, since the decryption values in both hybrids are exactly the same.

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

Indistinguishability follows from security of SKE as above.

$\mathcal{H}(j,2,3)$: Set $\mathsf{mode\text{-}trap}_1=1$ with target $T-1$, $\mathsf{mode\text{-}trap}_2=\perp$, and $\mathsf{mode\text{-}trap}_3=1$ with target $T-2$.

Indistinguishability follows from security of kFE, since the decryption values in both hybrids are exactly the same.

$\mathcal{H}(j, 2, 4)$: Switch normal root key to punctured key at position $T - 2$.

Indistinguishability follows from security of kFE as discussed above.

$\mathcal{H}(j, 2, 5)$: Switch randomness to true in the ciphertext hardwired in key.

Indistinguishability follows from security of cPRF as discussed above.

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

Indistinguishability follows from security of $\mathsf{1FE}$.

$\mathcal{H}(j, 2, 7)$: Switch randomness back to use the PRF in the ciphertext hardwired in key.

Indistinguishability follows from security of cPRF as discussed above.

$\mathcal{H}(j, 2, 8)$: Switch punctured root key to normal root key.

Indistinguishability follows from security of kFE as discussed above.

$\mathcal{H}(j, 3, 1)$: Intuitively, we slide the trapdoor left by one step, i.e. change target time-steps to $T - 2$ and $T - 3$ in the ciphertext. Now slot 1 for $T - 2$ corresponds to $b = 1$ and slot 2 for $T - 3$ to $b = 0$. Set $\mathsf{mode\text{-}real} = \mathsf{mode\text{-}trap}_3 = \perp$ and $\mathsf{mode\text{-}trap}_1 = \mathsf{mode\text{-}trap}_2 = 1$.

Indistinguishability follows from security of kFE, since the decryption values in both hybrids are exactly the same. Note that now slot $T - 1$ is redundant, since $T - 2$ ciphertext is already switched to $b = 1$.

Hybrid $\mathcal{H}(j, 3, i)$ will be analogous to $\mathcal{H}(j, 2, i)$ for $i \in [8]$.

As we proceed left in the execution chain one step at a time, we reach step ℓ where $\ell = |\mathbf{w}|$, i.e. time steps for which kFE ciphertexts are provided by the encryptor. At this point we will hardwire the Agg key instead for the symbol ciphertexts and the Next key for the state ciphertexts with the exception at time step 1 when we will hardwire both the symbol ciphertext and the start state ciphertext in Agg key itself.

After going through all the global salt values and all the key values, we replace the challenge ciphertext to have $\mathsf{mode\text{-}real} = 1$ and message corresponding to $b = 1$, one step at a time. This is analogous to the case of single input TMFE, except that we must additionally track global salt values.

$\mathcal{H}(T)$: In this hybrid all ciphertexts have $\mathsf{mode\text{-}real} = 1$, all other trapdoor information is set to $\perp$ and $b = 1$ is used. This is the real world with $b = 1$.

Indistinguishability from $\mathcal{H}(j+1, 1, 1)$ follows from security of kFE since the decryption values in both hybrids are exactly the same.

A.3 Constrained PRF for our Function Family

Our proof makes use of a set of delegatable constrained pseudorandom functions (cPRF). We require T delegatable cPRFs, denoted by F_i for $i \in [T]$, where each cPRF in turn supports T delegations. The sequence of delegated keys for F_i are denoted by $\{K_{i,t}\}_{i,t\in[T]}$, corresponding to functions $f_{i,t}$, such that the satisfying set of $f_{i,t+1}$ is strictly contained within the satisfying set of $f_{i,t}$, for all $i, t \in [T]$.

In more detail, for any polynomial $\mathsf{poly}(\lambda)$, define $f_{i,t} : \{0,1\}^{\lambda+\mathsf{poly}(\lambda)} \to \{0,1\}$ as follows.

$$\begin{aligned} f_{i,t}(x\|z) &= 1 \quad if \;\; x \geq t \;\wedge (x\|z) \neq i \\ &= 0 \quad otherwise \end{aligned}$$

Thus, the root key (and hence all delegated keys) of F_i are punctured at the point i.

Overview. We provide a construction for a cPRF F which supports puncturing and delegation as required; the T cPRFs F_i for $i \in [T]$ may each be constructed similarly. To begin, note that we require the root key of F to be punctured at a point i^* (say). The cPRF construction for punctured PRF [Boneh and Waters (2013); Kiayias *et al.* (2013); Boyle *et al.* (2014)] (which is in turn inherited from the standard PRG based GGM [Goldreich *et al.* (1986)]) immediately satisfies this constraint, so we are left with the question of delegation.

Recall that we are required to delegate T times, where T is the (polynomial) runtime of the Turing machine on the encrypted input (please see Section 2.7), and the j^{th}

delegated key must support evaluation of points $\{(k\|z) : z \in \{0,1\}^\lambda\}$ for $k \geq j$, *except* when $(k\|z) = i^*$. This may be viewed as the j^{th} key being punctured on points $[1, j-1] \cup i^*$. We show that the GGM based construction for puncturing a single point can be extended to puncturing an interval (plus an extra point). Intuitively, puncturing an interval corresponds to puncturing at most λ internal nodes in the GGM tree. In more detail, we show that regardless of the value of j, it suffices to puncture at most λ points in the GGM tree to achieve puncturing of the entire interval $[1, j-1]$.

Construction. Formally, the cPRF F is defined as follows. Our constrain algorithm takes as input the set of points on which to puncture the PRF, as opposed to the satisfying set. We compute the GGM tree as in Figure A.8 and number the leaves from 1 to $2^{(\lambda+\mathsf{poly}(\lambda))}$.

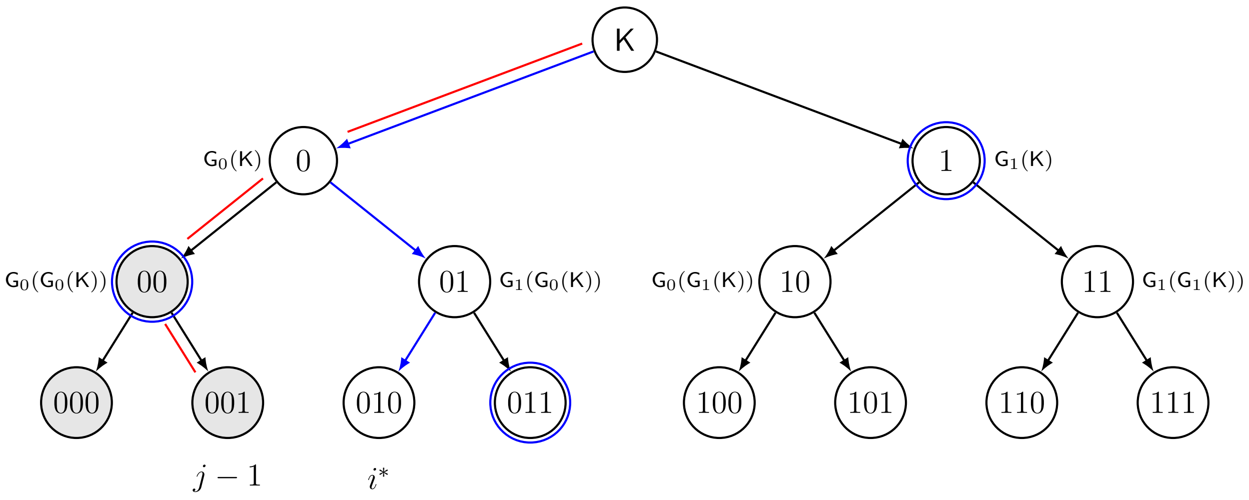

Figure A.8: To puncture $i^* = 010$ draw path from root to i^* and reveal nodes that are siblings along the path. To puncture interval $[1, 2] \cup \{i^*\} = \{000, 001\} \cup \{010\}$, compute the set Grey $= \{000, 001, 00\}$ and the punctured set $\mathcal{P} = \{00, 010\}$. Further compute the initial revealed set $\mathcal{R}_0 = \{(1, 01), (1, 00, 011)\}$ and replace 00 and 01 by 011 to get the final revealed set $\mathcal{R}_f = \{1, 011\}$.

$\mathsf{Setup}(1^\lambda)$: Sample a standard length doubling PRG G with seed s_0. Output $\mathsf{mpk} = G$ and $\mathsf{K}_0 = \mathsf{s}_0$. As usual, we will denote by G_0 the first half of the PRG output and by G_1 the second half.

$\mathsf{Constrain}(\mathsf{K}_0, [1, j-1] \cup i^*)$: Upon input the root key K_0 and the set of points to be punctured, do the following:

1. **Compute puncturing set $\mathcal{P}$:** Initialize $\mathcal{P}$ to contain the point i^*. Compute the path from the root node to node corresponding to point $j-1$. For any

right edge (a, b) along the path, mark the left child of a grey. Mark the final node $j-1$ grey. At this point we have a set of grey nodes which must be punctured. Minimize this set by checking whether both children of a node are grey, in which case, also mark the parent grey. Finally, add the grey nodes which do not have grey parents to a set $\mathcal{P}$.

2. **Computing revealed set $\mathcal{R}$:** For every node in the set $\mathcal{P}$, compute the punctured key (as in GGM) as follows. For every node $k \in \mathcal{P}$, compute the path from the root to k, and add the siblings of all nodes along the path to the set $\mathcal{R}$ [1]. Trim this set so as to remove conflicts caused by overlapping paths as follows: if any punctured node b in $\mathcal{P}$ is a descendent of some node a in $\mathcal{R}$, remove a from $\mathcal{R}$, compute the path from a to b and add all the siblings of the nodes on this path to $\mathcal{R}$. Repeat until there are no more changes to $\mathcal{R}$.
3. Output $\mathsf{K}_j = \mathcal{R}$.

$\mathsf{KeyDel}(\mathsf{K}_j, f_{j+1})$: Given the punctured key for set $[1, j]$, compute the punctured key for $[1, j+1]$ as follows. Note that it suffices to delegate from j to $j+1$ to imply delegation from j to any j' for $j' > j$.

1. Consider the case when j is a left child and $j+1$ is a right child of the same parent. In this case, the set K_j contains the node corresponding to $j+1$. Delete this node and return the resultant set as K_{j+1}.
2. Consider the case when j is a right child and $j+1$ is a left child of the neighbouring parent. In this case K_j contains the parent of node $j+1$. Use the parent to evaluate the value corresponding to node $j+2$, remove the parent and add the value corresponding to node $j+2$.

$\mathsf{Eval}(\mathsf{K}_j, y)$: Evaluate the GGM tree on input y as $G_{y_1} \circ \ldots \circ G_{y_n}(\mathrm{s}_0)$ and output it. Note that K_j contains enough information to compute the path from root to y as long as y is supported by K_j.

Correctness. We argue correctness of the $\mathsf{Constrain}$ algorithm first. To begin, we claim that to puncture the interval $[1, j-1]$, it suffices to compute the path from the root node to $j-1$, and puncture the left siblings of any right edges along the path, i.e. if (a, b) is a right edge along the path, we puncture the left child of a. Since a descendent of the left child of a must necessarily have value $< j-1$, it is necessary to puncture these nodes. Moreover it is sufficient, along with $j-1$ to puncture these nodes, because i) any node of value $< j-1$ must have an ancestor, say a_{j-1} which lies along the path P from root to $j-1$ ii) If $(a_{j-1}, b_{j-1}) \in P$ for some b_{j-1}, then (a_{j-1}, b_{j-1}) is a right edge. Since $\mathsf{Constrain}$ algorithm populates $\mathcal{P}$ with this set of points and then minimizes this

[1]Note that this is exactly the constrained key provided for a single punctured point in the GGM based construction.

set, we have that $\mathcal{P}$ represents the punctured points in the tree. Next, we argue that the nodes returned via the set $\mathcal{R}$ is correct: to see this, note that $\mathcal{R}$ is initially populated with all the constrained keys for each punctured point in $\mathcal{P}$, and this set is trimmed to remove conflicts caused by overlapping paths. Thus, the resultant nodes returned in the set $\mathcal{R}$ capture the intersection of points whose evaluation is admitted by each punctured key.

Finally, note that the $\mathsf{Constrain}$ algorithm runs in polynomial time: this is because we may use binary search to compute the path from the root to any node in the graph, and all operations deal with listing the siblings along these paths which take $O(\mathsf{poly}(\lambda))$. Moreover, we note that there is at most one punctured point at every level for any interval $[1, j-1]$, which implies that the total runtime of $\mathsf{Constrain}$ is $O((\mathsf{poly}(\lambda))^2)$.

Correctness of Eval is immediate, since evaluation is exactly the same as GGM evaluation. Correctness of KeyDel is also straightforward, since we only delegate one step at a time, hence it suffices to simply puncture one additional node corresponding to a point $j+1$, which is either the right child of the same parent as j, or the left child of the neighbouring parent. Puncturing a single node is immediate in either of these cases, as described in KeyDel above.

Security. We argue that given a punctured key, an adversary cannot distinguish a pseudorandom value from a random value on any input y that is not supported by the punctured key. Since by construction of the constrained key, the adversary does not possess any node along the path from the root to the node corresponding to y, we have that the node corresponding to y is pseudorandom by the standard hybrid argument for GGM security.

A.4 Constructing DI Secure Functional Encryption

Let $\mathsf{1FE}$ be a single input functional encryption scheme which satisfies standard indistinguishability based security. We will construct a single input functional encryption scheme DiFE satisfying distributional indistinguishability as shown below. Our proof follows the strategy of embedding a hidden thread in the functionality which is only active during simulation [Caro *et al.* (2013); Ananth *et al.* (2015*a*)] and therefore, uses an additional CPA-secure symmetric key encryption scheme $\mathsf{Sym} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

We note that the scheme presented below is public key, but directly lends itself to a private key version by instead relying on private key 1FE.

$\mathsf{DiFE.Setup}(1^\lambda, 1^n)$: Upon input the security parameter and length of input message, do the following:

1. Invoke $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, 1^{n+\lambda+1})$ and output $(\mathsf{PK}, \mathsf{MSK})$.

$\mathsf{DiFE.Enc}(\mathsf{PK}, \mathbf{x})$: Upon input the public key PK and a vector $\mathbf{x} \in \mathcal{X}^n$, do the following:

1. Output $\mathsf{CT}_\mathbf{x} = \mathsf{1FE.Enc}\big(\mathsf{PK}, (\mathbf{x}, \mathbf{0}, 0)\big)$.

$\mathsf{DiFE.KeyGen}(\mathsf{MSK}, f)$: Upon input the master secret key MSK and a circuit f, do the following:

1. Choose CT randomly from the space of Sym ciphertexts.
2. Output $\mathsf{SK}_f = \mathsf{1FE.KeyGen}(\mathsf{MSK}, f')$ where f' is as defined in Figure A.9.

Functionality $f'_{f,\mathsf{CT}}(\mathbf{x}, \mathsf{Sym.K}, \mathsf{mode})$
If $\mathsf{mode} = 0$, output $y = f(\mathbf{x})$ else output $y = \mathsf{Sym.Dec}(\mathsf{K}, \mathsf{CT})$.

Figure A.9: Functionality $f'_{f,\mathsf{CT}}$

$\mathsf{DiFE.Dec}(\mathsf{PK}, \mathsf{CT}_\mathbf{x}, \mathsf{SK}_f)$: Upon input the public key PK, a ciphertext $\mathsf{CT}_\mathbf{x}$ and a function key SK_f, compute $\mathsf{1FE.Dec}(\mathsf{PK}, \mathsf{CT}_\mathbf{x}, \mathsf{SK}_f)$ and output it.

Correctness. We have by correctness of 1FE that decryption recovers $f(\mathbf{x})$ as desired.

Proof of Security.

Next, we argue that the DiFE scheme constructed above is secure.

Theorem A.4.1. *Assume that* 1FE *is an* FE *scheme that satisfies standard indistinguishability based security and that* Sym *is a CPA-secure symmetric key encryption scheme. Then, the* DiFE *scheme constructed above satisfies selective, distributional indistinguishability based security.*

Proof. The proof proceeds via a sequence of hybrids where the first hybrid corresponds to an encryption of vector $\mathbf{x}_0$ chosen from distribution D_0 and the last hybrid corresponds to an encryption of vector $\mathbf{x}_1$ chosen from distribution D_1.

Hybrid 0: This is the real world with $\mathbf{x}_0 \leftarrow D_0$.

Hybrid 1: In this world, we hardwire the output of the function $y = f(\mathbf{x}_0)$, where $\mathbf{x}_0 \leftarrow D_0$ into the function key using symmetric key encryption. That is, let $\mathsf{Sym.K} \leftarrow \mathsf{Sym.Gen}(1^\lambda)$ and $\mathsf{CT} = \mathsf{Sym.Enc}(\mathsf{Sym.K}, \mathsf{y})$.

Hybrid 2: In this world, change the message in the ciphertext, i.e. message encoded is $(\perp, \mathsf{Sym.K}, \mathsf{mode} = 1)$.

Hybrid 3: In this world, we change the value of y to $y = f(\mathbf{x}_1)$.

Hybrid 4: In this world, we change the message encrypted to $(\mathbf{x}_1, \mathbf{0}, \mathsf{mode} = 0)$ where $\mathbf{x}_1 \leftarrow D_1$.

Hybrid 5: In this world, we change the value of CT hardwired in the key back to random.

Next, we argue that consecutive hybrids are indistinguishable.

Lemma A.4.2. *Hybrids* 0 *and* 1 *are indistinguishable assuming the security of* Sym.

Proof. The only thing that changes between Hybrid 0 and 1 is the choice of CT, so that in the former it is chosen randomly and in the latter case it is an honest encryption of the scheme Sym. Given an adversary $\mathcal{A}$ who distinguishes between Hybrids 0 and Hybrid 1, we construct an adversary $\mathcal{B}$ who breaks the semantic security of Sym.

$\mathcal{B}$ generates the public key honestly and returns it to $\mathcal{A}$. When $\mathcal{A}$ outputs two challenge distributions D_0, D_1, $\mathcal{B}$ samples $\mathbf{x}_0 \leftarrow D_0$. It honestly computes ciphertexts for $(\mathbf{x}_0, \mathbf{0}, \mathsf{mode} = 0)$ and returns these to $\mathcal{A}$. When $\mathcal{A}$ requests a function key f, $\mathcal{B}$ computes the value $y = f(\mathbf{x}_0)$, and sends y to the Sym challenger. The Sym challenger responds with CT which is either an honest encryption of y or an element chosen randomly from the ciphertext space. $\mathcal{B}$ uses CT in constructing the function key and returns this to $\mathcal{A}$. Now, if CT is random, $\mathcal{A}$ sees the view of Hybrid 0 and if it is an encryption of y, it sees the view of Hybrid 1. $\square$

Lemma A.4.3. *Hybrids* 1 *and* 2 *are indistinguishable assuming the security of* $\mathsf{1FE}$.

Proof. The only difference between Hybrids 1 and 2 is that in the former the encrypted message is $(\mathbf{x}_0, \mathbf{0}, \mathsf{mode} = 0)$ and in the latter it is $(\perp, \mathsf{Sym.K}, \mathsf{mode} = 1)$. Assume there is an adversary $\mathcal{A}$ who distinguishes between Hybrid 1 and Hybrid 2, we construct an adversary $\mathcal{B}$ who can break the security of $\mathsf{1FE}$.

$\mathcal{B}$ does the following:

1. It obtains the public key from the $\mathsf{1FE}$ challenger and returns this to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs two distribution pairs (D_0, D_1), it samples $\mathbf{x}_0 \leftarrow D_0$, $\mathsf{Sym.K}$ and returns challenges $(\mathbf{x}_0, \mathbf{0}, \mathsf{mode} = 0)$ and $(\perp, \mathsf{Sym.K}, \mathsf{mode} = 1)$ to the $\mathsf{1FE}$ challenger. It obtains an encryption of one of them chosen at random and returns this to $\mathcal{A}$.
3. When $\mathcal{A}$ outputs a function f, $\mathcal{B}$ constructs the function f' as described in Figure A.9 and sends this to the $\mathsf{1FE}$ challenger. Here, CT is computed as $\mathsf{Sym.Enc}(\mathsf{Sym.K}, \mathsf{y})$ where $y = f(\mathbf{x}_0)$. It returns the obtained key to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs a guess bit, it outputs the same.

When the $\mathsf{1FE}$ challenger returns an encryption of $(\mathbf{x}_0, \mathbf{0}, \mathsf{mode} = 0)$, $\mathcal{A}$ sees the view of Hybrid 1, and when it returns an encryption of $(\perp, \mathsf{Sym.K}, \mathsf{mode} = 1)$, it sees the view of Hybrid 2. Note that in either case the decrypted value is the same. Thus, the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. $\square$

Lemma A.4.4. *Hybrids* 2 *and* 3 *are indistinguishable since* $f(\mathbf{x}_0) \approx f(\mathbf{x}_1)$.

Proof. The only thing that differs in these two hybrids is the value of y. Given an adversary $\mathcal{A}$ who distinguishes between Hybrids 2 and 3, we construct an adversary $\mathcal{B}$ who distinguishes between $f(\mathbf{x}_0)$ and $f(\mathbf{x}_1)$. $\mathcal{B}$ does the following:

1. It samples the public key honestly and gives it to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs challenge distributions D_0 and D_1, it computes the ciphertext for $(\perp, \mathsf{Sym.K}, \mathsf{mode} = 1)$ honestly and returns it.
3. When $\mathcal{A}$ outputs a key request for function f, $\mathcal{B}$ outputs (D_0, D_1, f) to the distribution challenger. $\mathcal{B}$ receives $y_0 = f(\mathbf{x}_0)$ or $y_1 = f(\mathbf{x}_1)$, where $\mathbf{x}_b \leftarrow D_b$ for $b \in \{0, 1\}$. It uses this to construct the circuit f'. It then computes the key for f' honestly and returns this to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs a guess, $\mathcal{B}$ outputs the same.

If $\mathcal{B}$ receives y_0, $\mathcal{A}$ sees the distribution of Hybrid 2, else it sees the distribution of Hybrid 3. The advantage of $\mathcal{A}$ therefore translates to an advantage of $\mathcal{B}$. $\square$

Lemma A.4.5. *Hybrids* 3 *and* 4 *are indistinguishable assuming the security of* $\mathsf{1FE}$.

Proof. The only difference between Hybrids 3 and 4 is that in the former, the message encoded in the ciphertext is $(\perp, \mathsf{Sym.K}, \mathsf{mode} = 1)$ and in the latter the message encrypted is $(\mathbf{x}_1, \mathbf{0}, \mathsf{mode} = 0)$. Note that in both cases, we have the same output of decryption hence the two ciphertexts are indistinguishable by security of $\mathsf{1FE}$. $\square$

Lemma A.4.6. *Hybrids 4 and 5 are indistinguishable assuming the security of* Sym.

Proof. The proof is similar to Lemma A.4.2. $\square$

$\square$

A.5 Constructing Decomposable Functional Encryption for Circuits

Given any single-input circuit FE scheme $\mathsf{1FE}$ satisfying standard indistinguishability based security, a *projective* garbled circuit scheme $\mathsf{GC} = (\mathsf{GCirc}, \mathsf{GInp}, \mathsf{GEval})$ with indistinguishability based security [Jafargholi *et al.* (2017)] supporting a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ with n-bit inputs, a simple PRF $\mathsf{F} = (\mathsf{F.Setup}, \mathsf{F.Eval})$ and a symmetric encryption scheme SYM, we can construct a single-input decomposable FE scheme DFE supporting the circuit class $\mathcal{C}$[2]. We note that projective garbled circuit schemes satisfying indistinguishability based security are implied from one-way functions [Jafargholi *et al.* (2017)]. The intuition behind the construction is as follows.

Intuition: The public key and master secret key for DFE would be the same as that of $\mathsf{1FE}$. Given an n-bit message $\mathbf{x} = (x_1, \ldots, x_n)$, the DFE encryption algorithm samples a PRF key K and generates n $\mathsf{1FE}$ ciphertexts encoding (K, i, x_i). DFE key generation takes the master secret key and a circuit C as input and generates a secret key for a circuit $\widehat{C}_{C,\mathsf{salt}}$. The circuit $\widehat{C}_{C,\mathsf{salt}}$ takes a $\mathsf{1FE}$ message (K, i, x_i) as input and generates a garbled circuit $\widetilde{C}$ corresponding to C and a garbled input label for the ith bit x_i using randomness $\mathsf{PRF}(\mathsf{K}, \mathsf{salt})$. This relies on the projective property of GC [Jafargholi *et al.* (2017)], i.e., each bit of the garbled input $\widetilde{\mathbf{x}}$ only depends on one bit of the actual input $\mathbf{x}$. For decryption, DFE runs the $\mathsf{1FE}$ decryption on all the n $\mathsf{1FE}$ ciphertexts to obtain the

[2]We thank Vinod Vaikuntanathan for suggesting the generic transformation from FE to decomposable FE.

garbled circuit $\widetilde{C}$ and the garbled input $\widetilde{\mathbf{x}}$ and then evaluates the garbled circuit to get the output $C(\mathbf{x})$.

For proving security, we additionally need to rely on a symmetric key scheme following standard techniques employing trapdoor modes from [Caro *et al.* (2013); Ananth *et al.* (2015*a*)]. The details follow as shown below.

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

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

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

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

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

1. Sample a random $\mathsf{salt} \leftarrow \{0,1\}^\lambda$, $\mathsf{CT}_i \leftarrow \{0,1\}^{\ell(\lambda)}, \forall i \in [0, n]$.
2. Output $\mathsf{SK}_{\widehat{C}} = \mathsf{1FE.KeyGen}(\mathsf{MSK}, \widehat{C}_{C,\mathsf{salt},\{\mathsf{CT}_i\}_{i\in[n]},\mathsf{CT}_0})$, where the circuit $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$ is a circuit described in Figure A.10.

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

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

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

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

(a) Initialize the vector $\mathsf{out} = (c_1, c_2)$, where $c_j = \perp, \forall j \in [2]$.

(b) If $\mathsf{mode} = 1$, do the following:
   i. Let $\mathsf{out}.c_1 = \mathsf{SYM.Dec}(\mathsf{SYM.K}, \mathsf{SYM.CT}_i)$.
   ii. If $i = 1$, let $\mathsf{out}.c_2 = \mathsf{SYM.Dec}(\mathsf{SYM.K}, \mathsf{SYM.CT}_{\widetilde{C}})$.

(c) If $\mathsf{mode} = 0$, do the following:
   i. Compute randomness $r = \mathsf{F.Eval}(\mathsf{K}, \mathsf{salt})$.
   ii. Use randomness r to generate the garbled circuit for C as $(\widetilde{C}, \mathsf{sk}) = \mathsf{GCirc}(1^\lambda, C; r)$ as well as to generate the label corresponding to the i^{th} input wire as $\ell_{i,x_i} = \mathsf{GInp}(\mathsf{sk}, (x_i, i); r)$.
   iii. Let $\mathsf{out}.c_1 = \ell_{i,x_i}$. If $i = 1$, let $\mathsf{out}.c_2 = \widetilde{C}$.

(d) **Output :** out.

Figure A.10: Functionality $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$

Proof of Security.

Next, we argue that the DFE scheme constructed above is secure.

Theorem A.5.1. *Assume that* 1FE *is an* FE *scheme satisfying standard indistinguishability based security,* GC *is a projective garbling scheme for circuits satisfying indistinguishability based security,* Sym *is a secure symmetric key encryption scheme and* F *is a secure* PRF. *Then, the* DFE *scheme constructed above is a single-input, decomposable* FE *scheme satisfying selective indistinguishability based security.*

Proof. The proof proceeds via a sequence of hybrids where the first hybrid corresponds to an encryption of message $\mathbf{x}_0 \in \{0,1\}^n$ and the last hybrid corresponds to an encryption of message $\mathbf{x}_1 \in \{0,1\}^n$.

Hybrid 0: This is the real world with message $\mathbf{x}_0 = (x_1^0, \ldots, x_n^0) \in \{0,1\}^n$.

Hybrid 1: In this world, we hardwire $\widehat{C}$ with its output, namely the garbled circuit $(\widetilde{C}, \mathsf{sk})$ and input labels $\{\ell_{i,x_i^0}\}_{i\in[n]}$, using symmetric key encryption.

Hybrid 2: In this world, we change the message in each of the n 1FE ciphertexts from $(\mathsf{K}, \mathbf{0}, i, x_i^0, 0)$ to $(\perp, \mathsf{Sym.K}, i, \perp, 1)$, i.e., the message encoded in CT_{x_i} is $(\perp, \mathsf{Sym.K}, i, \perp, \mathsf{mode} = 1), \forall i \in [n]$.

Hybrid 3: In this world, we use true randomness to generate the garbled circuit and garbled inputs instead of using randomness generated by PRF. Everything else remains the same as that of the previous hybrid. Note that the garbled input labels $\{\ell_{i,x_i^0}\}_{i\in[n]}$ encoded by $\{\mathsf{SYM.CT}_i\}_{i\in[n]}$ correspond to the input message bits of $\mathbf{x}_0 = (x_1^0, \ldots, x_n^0)$ from the previous hybrids.

Hybrid 4: In this world, we change the garbled input labels to $\{\ell_{i,x_i^1}\}_{i\in[n]}$ corresponding to the input message bits of $\mathbf{x}_1 = (x_1^1, \ldots, x_n^1)$ encoded by $\{\mathsf{SYM.CT}_i\}_{i\in[n]}$ and hardwired in the key for $\widehat{C}$.

Hybrid 5: In this world, we change the true randomness back to randomness generated by the PRF for computing the garbled circuit and garbled inputs. Everything else remains the same as that of the previous hybrid. Note that the garbled input labels $\{\ell_{i,x_i^1}\}_{i\in[n]}$ encoded by $\{\mathsf{SYM.CT}_i\}_{i\in[n]}$ now correspond to the input message bits of $\mathbf{x}_1 = (x_1^1, \ldots, x_n^1)$ from the previous hybrid.

Hybrid 6: In this world, we change the message in each of the n $\mathsf{1FE}$ ciphertexts from $(\perp, \mathsf{Sym.K}, i, \perp, 1)$ to $(\mathsf{K}, \mathbf{0}, i, x_i^1, 0)$, i.e., the message encoded in CT_{x_i} now is $(\mathsf{K}, \mathbf{0}, i, x_i^1, 0), \forall i \in [n]$.

Hybrid 7: In this world, we change the hardwired values in $\widehat{C}$ corresponding to the $\{\mathsf{SYM.CT}_i\}_{i\in[n]}$ and $\mathsf{SYM.CT}_{\widehat{C}}$ slots back to random strings from the ciphertext space of SYM. Note that this corresponds to the real world with message $\mathbf{x}_1 = (x_1^1, \ldots, x_n^1) \in \{0,1\}^n$.

Next, we argue that consecutive hybrids are indistinguishable.

Lemma A.5.2. *Hybrids* 0 *and* 1 *are indistinguishable assuming the security of* SYM.

Proof. The only thing that changes between Hybrids 0 and 1 are the choices of $\{\mathsf{SYM.CT}_i\}_{i\in[0,n]}$, so that in the former it is chosen randomly and in the latter case it is an honest encryption of the scheme SYM. Given an adversary $\mathcal{A}$ which distinguishes between Hybrid 0 and Hybrid 1, we construct an adversary $\mathcal{B}$ which breaks the semantic security of SYM. $\mathcal{B}$ does the following:

1. $\mathcal{B}$ generates $(\mathsf{PK}, \mathsf{MSK}) = (\mathsf{1FE.PK}, \mathsf{1FE.MSK}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, 1^{2\lambda+\log n+2})$ honestly and returns PK to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs a pair of challenge messages $(\mathbf{x}_0, \mathbf{x}_1)$, $\mathcal{B}$ samples a PRF key $\mathsf{K} \leftarrow \mathsf{F.Setup}(1^\lambda)$ and honestly computes $\mathsf{CT}_{x_i} = \mathsf{1FE.Enc}(\mathsf{PK}, (\mathsf{K}, \mathbf{0}, i, x_i^0, \mathsf{mode} = 0)), \forall i \in [n]$ and returns $\mathsf{CT}_{\mathbf{x}} = \{\mathsf{CT}_{x_i}\}_{i \in [n]}$ to $\mathcal{A}$.

3. When $\mathcal{A}$ requests a function key for C, $\mathcal{B}$ samples $\mathsf{salt} \leftarrow \{0,1\}^\lambda$ and computes $r = \mathsf{F.Eval}(\mathsf{K}, \mathsf{salt})$. It then generates the garbled circuit $(\widetilde{C}, \mathsf{sk}) = \mathsf{GCirc}(1^\lambda, C; r)$ and the input labels $\{\ell_{i,x_i^0} = \mathsf{GInp}(\mathsf{sk}, (i, x_i); r)\}_{i \in [n]}$ honestly. $\mathcal{B}$ then sends $\big(\{\ell_{i,x_i^0}\}_{i\in[n]}, \widetilde{C}\big)$ to the SYM challenger. The SYM challenger responds with $\big(\{\mathsf{SYM.CT}_i\}_{i\in[n]}, \mathsf{SYM.CT}_{\widetilde{C}}\big)$ upon which $\mathcal{B}$ constructs $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$ and generates a secret key $\mathsf{SK}_{\widehat{C}} = \mathsf{1FE.KeyGen}\big(\mathsf{MSK}, \widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}\big)$ honestly. $\mathcal{B}$ sends $\mathsf{SK}_{\widehat{C}}$ to $\mathcal{A}$.

4. When $\mathcal{A}$ outputs a guess bit, it outputs the same.

Now, $\mathcal{A}$ sees the view of Hybrid 0 if $\big(\{\mathsf{SYM.CT}_i\}_{i\in[n]}, \mathsf{SYM.CT}_{\widetilde{C}}\big)$ is random and $\mathcal{A}$ sees the view of Hybrid 1 if $\big(\{\mathsf{SYM.CT}_i\}_{i\in[n]}, \mathsf{SYM.CT}_{\widetilde{C}}\big)$ is an encryption of $\big(\{\ell_{i,x_i^0}\}_{i\in[n]}, \widetilde{C}\big)$. □

Lemma A.5.3. *Hybrids* 1 *and* 2 *are indistinguishable assuming the security of* $\mathsf{1FE}$.

Proof. The only difference between Hybrids 1 and 2 is that in the former the encrypted messages are $\{(\mathsf{K}, \mathbf{0}, i, x_i^0, \mathsf{mode} = 0)\}_{i\in[n]}$ and in the latter as $\{(\bot, \mathsf{SYM.K}, i, \bot, \mathsf{mode} = 1)\}_{i\in[n]}$. Assuming there is an adversary $\mathcal{A}$ which distinguishes between Hybrid 1 and Hybrid 2, we construct an adversary $\mathcal{B}$ which breaks the security of $\mathsf{1FE}$.

$\mathcal{B}$ does the following:

1. It obtains the public key PK from the $\mathsf{1FE}$ challenger and returns this to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs a pair of challenge messages $(\mathbf{x}_0, \mathbf{x}_1)$, it samples $\mathsf{K} \leftarrow \mathsf{F.Setup}(1^\lambda)$, a symmetric encryption key $\mathsf{SYM.K}$ and then returns n $\mathsf{1FE}$ challenge message pairs $\{(\mathsf{K}, \mathbf{0}, i, x_i^0, \mathsf{mode} = 0)\ , (\bot, \mathsf{SYM.K}, i, \bot, \mathsf{mode} = 1)\}_{i\in[n]}$ w.l.o.g. to the $\mathsf{1FE}$ challenger. It obtains $\mathsf{CT}_{\mathbf{x}} = \{\mathsf{CT}_{x_i}\}_{i\in[n]}$ and returns this to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a function query for C, $\mathcal{B}$ constructs $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$ as described in Figure A.10 and sends this to the $\mathsf{1FE}$ challenger. Here, $\mathsf{SYM.CT}_{\widetilde{C}}$ is computed as $\mathsf{SYM.Enc}(\mathsf{SYM.K}, \widetilde{C})$ where $(\widetilde{C}, \mathsf{sk}) = \mathsf{GCirc}(1^\lambda, C; \mathsf{F.Eval}(\mathsf{K}, \mathsf{salt}))$ while $\mathsf{SYM.CT}_i$ are computed as $\mathsf{SYM.Enc}(\mathsf{SYM.K}, \ell_{i,x_i^0})$ where $\ell_{i,x_i^0} = \mathsf{GInp}(\mathsf{sk}, (i, x_i^0); \mathsf{F.Eval}(\mathsf{K}, \mathsf{salt})), \forall i \in [n]$. It returns the obtained key $\mathsf{SK}_{\widehat{C}}$ to $\mathcal{A}$.

4. When $\mathcal{A}$ outputs a guess bit, it outputs the same.

When the $\mathsf{1FE}$ challenger returns encryptions of $\{(\mathsf{K}, \mathbf{0}, i, x_i^0, \mathsf{mode} = 0)\}_{i\in[n]}$, $\mathcal{A}$ sees the view of Hybrid 1, and when it returns an encryption of $\{(\bot, \mathsf{SYM.K}, i, \bot, \mathsf{mode} = 1)\}_{i\in[n]}$, it sees the view of Hybrid 2. Note that in either case the decrypted value is the same and thus the reduction $\mathcal{B}$ is a valid 1FE adversary. Thus, the advantage of $\mathcal{A}$ translates to the advantage of $\mathcal{B}$. $\square$

Lemma A.5.4. *Hybrids* 2 *and* 3 *are indistinguishable assuming the security of* PRF F.

Proof. The only difference in Hybrid 2 from Hybrid 3 is that instead of randomness generated by the PRF, true randomness is used now to generate the garbled circuit and garbled input. Note that the PRF key is not explicitly needed in the Hybrid 2. Thus, assuming there is an adversary $\mathcal{A}$ which distinguishes between Hybrid 2 and Hybrid 3, we construct an adversary $\mathcal{B}$ which breaks the security of PRF F. $\mathcal{B}$ does the following:

1. $\mathcal{B}$ generates $(\mathsf{PK}, \mathsf{MSK}) = (\mathsf{1FE.PK}, \mathsf{1FE.MSK}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, 1^{2\lambda+\log n+2})$ honestly and returns PK to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs a pair of challenge messages $(\mathbf{x}_0, \mathbf{x}_1)$, $\mathcal{B}$ samples SYM.K and simulates the challenge message as $\mathsf{CT}_{\mathbf{x}} = \{\mathsf{CT}_{x_i}\}_{i\in[n]}$ where $\mathsf{CT}_{x_i} = \mathsf{1FE.Enc}(\mathsf{PK}, (\perp, \mathsf{SYM.K}, i, \perp, 1))\}_{i\in[n]}$.
3. When $\mathcal{A}$ outputs a function query for C, $\mathcal{B}$ first queries the PRF challenger upon which it receives r. It then uses r to compute the garbled circuit $(\widetilde{C}, \mathsf{sk}) = \mathsf{GCirc}(1^\lambda, C; r)$ as well as the garbled input labels $\ell_{i,x_i^0} = \mathsf{GInp}(\mathsf{sk}, (i, x_i^0); r) \forall i \in [n]$, honestly. $\mathcal{B}$ then samples salt $\leftarrow \{0,1\}^\lambda$ and computes $\{\mathsf{SYM.CT}_i = \mathsf{SYM.Enc}(\mathsf{SYM.K}, \ell_{i,x_i^0})\}_{i\in[n]}$ and $\mathsf{SYM.CT}_{\widetilde{C}} = \mathsf{SYM.Enc}(\mathsf{SYM.K}, \widetilde{\mathsf{C}})$. It then computes $\mathsf{SK}_{\widehat{C}} = \mathsf{1FE.KeyGen}(\mathsf{MSK}, \widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}})$ for the function $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$ as described in Figure A.10 and returns $\mathsf{SK}_{\widehat{C}}$ to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs a guess bit, $\mathcal{B}$ outputs the same.

If $\mathcal{B}$ had received $r = \mathsf{F.Eval}(\mathsf{K}, \mathsf{salt})$ from the the PRF challenger, $\mathcal{A}$ sees the distribution of Hybrid 2, else it sees the distribution of Hybrid 3 if r was sampled uniformly at random by the PRF challenger. The advantage of $\mathcal{A}$ therefore translates to an advantage of $\mathcal{B}$. $\square$

Lemma A.5.5. *Hybrids* 3 *and* 4 *are indistinguishable assuming the security of* GC.

Proof. The only difference between Hybrids 3 and 4 is that in the former, the messages encoded in $\{\mathsf{SYM.CT}_i\}_{i\in[n]}$ ciphertexts hardwired in $\widehat{C}$ were $\{\ell_{i,x_i^0}\}_{i\in[n]}$ while in the later, the encoded messages are $\{\ell_{i,x_i^1}\}_{i\in[n]}$. Note that in both cases, we have the same output of decryption since $C(\mathbf{x}_0) = C(\mathbf{x}_1)$ and hence the two hybrids are indistinguishable by indistinguishability based security of GC. More formally, we show that if there is an adversary $\mathcal{A}$ which distinguishes between Hybrid 3 and Hybrid 4, we construct an adversary $\mathcal{B}$ which breaks the security of GC. $\mathcal{B}$ does the following:

1. $\mathcal{B}$ generates $(\mathsf{PK}, \mathsf{MSK}) = (\mathsf{1FE.PK}, \mathsf{1FE.MSK}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, 1^{2\lambda+\log n+2})$ honestly and returns PK to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs a pair of challenge messages $(\mathbf{x}_0, \mathbf{x}_1)$, $\mathcal{B}$ samples $\mathsf{SYM.K}$ and simulates the challenge message as $\mathsf{CT}_{\mathbf{x}} = \{\mathsf{CT}_{x_i}\}_{i\in[n]}$ where $\mathsf{CT}_{x_i} = \mathsf{1FE.Enc}(\mathsf{PK}, (\perp, \mathsf{SYM.K}, i, \perp, 1))\}_{i\in[n]}$.

3. When $\mathcal{A}$ outputs a function query for C, $\mathcal{B}$ first constructs and sends the challenge message pair $((C, \mathbf{x}_0), (C, \mathbf{x}_1))$ to the GC challenger. On receiving $(\widetilde{C}, \widetilde{\mathbf{x}} = \{\ell_{i,x_i}\}_{i\in[n]})$ from the GC challenger, $\mathcal{B}$ computes $\{\mathsf{SYM.CT}_i = \mathsf{SYM.Enc}(\mathsf{SYM.K}, \ell_{i,x_i})\}_{i\in[n]}$ and $\mathsf{SYM.CT}_{\widetilde{C}} = \mathsf{SYM.Enc}(\mathsf{SYM.K}, \widetilde{\mathsf{C}})$. It then samples $\mathsf{salt} \leftarrow \{0,1\}^\lambda$ and generates a function key for the function $\widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}}$ as $\mathsf{SK}_{\widehat{C}} = \mathsf{1FE.KeyGen}(\mathsf{MSK}, \widehat{C}_{C,\mathsf{salt},\{\mathsf{SYM.CT}_i\}_{i\in[n]},\mathsf{SYM.CT}_{\widetilde{C}}})$. $\mathcal{B}$ returns $\mathsf{SK}_{\widehat{C}}$ to $\mathcal{A}$.

4. When $\mathcal{A}$ outputs a guess bit, $\mathcal{B}$ outputs the same.

Note that since $\mathcal{A}$ is a valid DFE adversary satisfying $C(\mathbf{x}_0) = C(\mathbf{x}_1)$, this implies $\mathcal{B}$ is a valid GC adversary. Further, if the GC challenger had returned $(\widetilde{C}, \widetilde{\mathbf{x}} = \{\ell_{i,x_i^0}\}_{i\in[n]})$, then $\mathcal{A}$ sees the view of Hybrid 3 and if the GC challenger had returned $(\widetilde{C}, \widetilde{\mathbf{x}} = \{\ell_{i,x_i^1}\}_{i\in[n]})$, then $\mathcal{A}$ sees the view of Hybrid 4. The advantage of $\mathcal{A}$ therefore translates to an advantage of $\mathcal{B}$. □

Lemma A.5.6. *Hybrids* 4 *and* 5 *are indistinguishable assuming the security of* PRF F.

Proof. The proof is similar to Lemma A.5.4. □

Lemma A.5.7. *Hybrids* 5 *and* 6 *are indistinguishable assuming the security of* 1FE.

Proof. The proof is similar to Lemma A.5.3. □

Lemma A.5.8. *Hybrids* 6 *and* 7 *are indistinguishable assuming the security of* SYM.

Proof. The proof is similar to Lemma A.5.2. □

□

A.5.1 Decomposable Functional Encryption for Circuits: Instantiations

We note that most functional encryption schemes in the literature are already decomposable, since a long input $\mathbf{x}$ is typically encoded bit by bit, using a separate public key component. Indeed, we do not know of any exception in the literature. For instance, recall the ciphertext of [Ananth *et al.* (2015*a*)]:

$$\begin{aligned}\mathsf{CT}_0 &\leftarrow \mathsf{OneCT.Enc}(\mathsf{OneCT.SK}, \mathbf{x}) \text{ and}\\ \mathsf{CT}_1 &\leftarrow \mathsf{Sel.Enc}(\mathsf{Sel.MPK}, (\mathsf{OneCT.SK}, \mathsf{K}, 0^\lambda, 0)).\end{aligned}$$

Above, CT_1 is a ciphertext component which is independent of the message (and depends only on randomness), hence it may be denoted as $\mathsf{CT}_{\mathsf{indpt}}$ in the notation above. Therefore, it remains to show that CT_0 is decomposable. This depends on the particular OneCT scheme that is chosen, but for instance, it was shown in [Ananth and Sahai (2016)] that the OneCT succinct FE scheme from LWE constructed by [Goldwasser *et al.* (2013*a*)] is decomposable. We refer the reader to [Ananth and Sahai (2016)] for details.

We note that the recent constructions of FE from constant degree multilinear maps [Lin (2017); Lin and Tessaro (2017)] also satisfy decomposability, despite the fact that they precompute high degree monomials which are encoded. To see this, note that the encrypt algorithm in [Lin (2017); Lin and Tessaro (2017)] takes as input a message $\mathbf{x}$ and *chooses* a PRG seed $\mathbf{s}$ (say) represented as a matrix. The encryptor computes a long message $\mathbf{y}$ (say) that consists of monomials computed over $\mathbf{x}, \mathbf{s}$. While the computation of arbitrary monomials would violate decomposability, in the above constructions, the monomials are *linear* in bits of $\mathbf{x}$, and the high degree terms are all computed over the bits of the seed $\mathbf{s}$. Our construction requires that the bits corresponding to the symbol and state of a TM be encoded separately, and these would form the input $\mathbf{x}$ in the constructions of [Lin (2017); Lin and Tessaro (2017)]. Intuitively, the PRG seed is used to derive randomness meant for computing a randomized encoding and is chosen independently of the input message $\mathbf{x}$. Hence, the constructions of [Lin (2017); Lin and Tessaro (2017)] also satisfy decomposability required by our compilers.

Next, we sketch how the construction of [Garg *et al.* (2013*a*)] can be seen to satisfy decomposability, with minor modifications. The ciphertext for a single bit message m in this scheme is (e_1, e_2, π), where $e_1 = \mathsf{Enc}(\mathsf{PK}_1, m)$ and $e_2 = \mathsf{Enc}(\mathsf{PK}_2, m)$ and π is a NIZK proof that e_1 and e_2 both encrypt the same bit. Note that here the two ciphertexts e_1 and e_2 are using distinct public key encryption schemes (i.e. these are not ciphertext components in decomposable FE). To argue decomposability, consider message $\mathbf{m} = (m_1, \ldots, m_n)$ as a vector of n bits rather than a single bit. Then, we may compute the encryptions bit by bit, and also test equality bit by bit in the NIZK, tying together all bits of m by common randomness, satisfying the given definition of decomposability.

In more detail, we may compute $e_1 = (e_{1,1}, \ldots, e_{1,n})$ and $e_2 = (e_{2,1}, \ldots, e_{2,n})$ as well as $e^* = \mathsf{Enc}(\mathsf{PK}_3, \mathsf{R})$ where:

- $\forall i \in [n]$, $e_{1,i}$ and $e_{2,i}$ encode message (m_i, R) where R is shared across all i. Note that R here is part of the encoded message (the encryption randomness used to construct the ciphertexts $e_{1,i}$ and $e_{2,i}$ is different and not denoted here).
- Denote by π_i the NIZK proof that $e_{1,i}$ and $e_{2,i}$ encode the same bit m_i and that $e_{1,i}, e_{2,i}$ encode the same R as e^*.

Then, the n ciphertext components of the decomposable FE are $(e_{1,1}, e_{2,1}, \pi_1), \ldots, (e_{1,n}, e_{2,n}, \pi_n)$ and the independent ciphertext component is e^* ($\mathsf{CT}_{\mathsf{indpt}}$ from Definition 2.6.2). Note that if an attacker tried to replace any one piece in this set, the R would not match (except with negligible probability) and the NIZK proof would not validate.

The proof of security is similar to [Garg *et al.* (2013*a*)].

APPENDIX B

Appendices for Chapter 3

B.1 Definitions: Predicate and Functional Encryption

B.1.1 Predicate and Bounded Key Functional Encryption for Circuits

We present the definition of predicate and bounded key functional encryption for general circuits similarly to [Gorbunov *et al.* (2015); Goldwasser *et al.* (2013*a*); Agrawal (2017)]. We follow the notation of [Agrawal (2017)] which provides a single definition for predicate encryption and succinct bounded key functional encryption. Since this primitive interpolates predicate and functional encryption, we denote it by PE^+. We note that this definition achieves input privacy but not machine privacy.

For $\lambda \in \mathbb{N}$, let $\mathcal{C}_{\mathsf{inp},\mathsf{d}}$ denote a family of circuits with inp bit inputs, an a-priori bounded depth d, and binary output and $\mathcal{C} = \{\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}\}_{\lambda\in\mathbb{N}}$. A predicate encryption scheme PE^+ for $\mathcal{C}$ over a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of four algorithms:

- $\mathsf{PE}^+.\mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}})$ is a PPT algorithm takes as input the unary representation of the security parameter, the length $\mathsf{inp} = \mathsf{inp}(\lambda)$ of the input and the depth $\mathsf{d} = \mathsf{d}(\lambda)$ of the circuit family $\mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}$ to be supported. It outputs the master public key and the master secret key $(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk})$.

- $\mathsf{PE}^+.\mathsf{Enc}(\mathsf{PE}^+.\mathsf{mpk}, \mathbf{x}, m)$ is a PPT algorithm that takes as input the master public key $\mathsf{PE}^+.\mathsf{mpk}$, a string $\mathbf{x} \in \{0,1\}^{\mathsf{inp}}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext $\mathsf{PE}^+.\mathsf{ct}$.

- $\mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}, C)$ is a PPT algorithm that takes as input the master public key $\mathsf{PE}^+.\mathsf{mpk}$, master secret key $\mathsf{PE}^+.\mathsf{msk}$, and a circuit $C \in \mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}$ and outputs a corresponding secret key $\mathsf{PE}^+.\mathsf{sk}_C$.

- $\mathsf{PE}^+.\mathsf{Dec}(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{sk}_C, C, \mathsf{PE}^+.\mathsf{ct})$ is a deterministic algorithm that takes as input the master public key $\mathsf{PE}^+.\mathsf{mpk}$, the secret key $\mathsf{PE}^+.\mathsf{sk}_C$, its associated circuit C, and a ciphertext $\mathsf{PE}^+.\mathsf{ct}$ and outputs either a message m' or $\perp$.

Definition B.1.1 (Correctness)**.** A predicate encryption scheme for circuits PE^+ is correct if for all $\lambda \in \mathbb{N}$, polynomially bounded inp and d, all circuits $C \in \mathcal{C}_{\mathsf{inp}(\lambda),\mathsf{d}(\lambda)}$, all $\mathbf{x} \in \{0,1\}^{\mathsf{inp}}$ such that $C(\mathbf{x}) = 1$ and for all messages $m \in \mathcal{M}$,

$$\Pr\left[\begin{array}{l} (\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}) \leftarrow \mathsf{PE}^+.\mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}), \\ \mathsf{PE}^+.\mathsf{ct} \leftarrow \mathsf{PE}^+.\mathsf{Enc}(\mathsf{PE}^+.\mathsf{mpk}, \mathbf{x}, m), \\ \mathsf{PE}^+.\mathsf{sk}_C \leftarrow \mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}, C) : \\ \mathsf{PE}^+.\mathsf{Dec}\Big(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{sk}_C, C, \mathsf{PE}^+.\mathsf{ct}\Big) \neq m \end{array}\right] = \mathrm{negl}(\lambda)$$

where the probability is taken over the coins of $\mathsf{PE}^+.\mathsf{Setup}$, $\mathsf{PE}^+.\mathsf{KeyGen}$, and $\mathsf{PE}^+.\mathsf{Enc}$.

Security. Next, we define simulation based security for PE^+. Note that simulation based security is impossible for functional encryption against an adversary that requests even a *single* key after seeing the challenge ciphertext [Boneh *et al.* (2011)], or an unbounded number of keys before seeing the challenge ciphertext [Agrawal *et al.* (2013)]. However, against an adversary who only requests an a-priori bounded number of keys *before* seeing the challenge ciphertext, simulation based security is possible but causes the ciphertext size to grow with the number of requested keys [Agrawal *et al.* (2013)].

Following [Agrawal (2017)], we provide a definition that subsumes single (or bounded) key functional encryption as well as predicate encryption; namely where an attacker can make an unbounded number of function queries C_i so long as it holds that the function keys do not decrypt the challenge ciphertext $\mathsf{PE}^+.\mathsf{Enc}(\mathsf{PE}^+.\mathsf{mpk}, \mathbf{x}, m)$ to recover m. Thus, it holds that $C_i(\mathbf{x}) = 0$ for all requested C_i. We shall refer to such C_i as 0-keys, and any C such that $C(\mathbf{x}) = 1$ as a 1-key. In our definition, the adversary can request a single arbitrary (i.e. 0 or 1) key followed by an unbounded polynomial number of 0-keys. As in [Agrawal (2017)], we refer to this security notion as $(1, \mathrm{poly})$ simulation security.

Definition B.1.2 ($(1, \mathrm{poly})$-Sel-SIM Security)**.** Let PE^+ be a predicate encryption scheme for a Boolean circuit family $\mathcal{C}$. For a stateful PPT adversary A and a stateful PPT simulator Sim, consider the following two experiments:

```
$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{PE^+},\mathsf{A}}(1^\lambda)}$:

1: $(X, \mathsf{Msg}, C^*) \leftarrow \mathsf{A}(1^\lambda)$
2: $(\mathsf{PE^+.mpk}, \mathsf{PE^+.msk}) \leftarrow \mathsf{PE^+.Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}})$
3: For $\mathbf{x}_i \in X$, let $b_i = m_i$.
4: Let $\mathsf{CT}_X :=$
       $\{\mathsf{PE^+.ct}_{\mathsf{x_i}} \leftarrow \mathsf{PE^+.Enc}\big(\mathsf{PE^+.mpk}, \mathsf{x_i}, \mathsf{b_i}\big)\}_{\mathsf{i}\in[|\mathsf{X}|]}$
5: $\mathsf{PE^+.sk}_{C^*} \leftarrow \mathsf{PE^+.KeyGen}(\mathsf{PE^+.mpk}, \mathsf{PE^+.msk}, C^*)$
6: $\alpha \leftarrow \mathsf{A}^{\mathsf{PE^+.KeyGen}(\mathsf{PE^+.mpk},\mathsf{PE^+.msk},\cdot)}(\mathsf{CT}_X, \mathsf{PE^+.sk}_{C^*})$
7: Output $(X, \mathsf{Msg}, \alpha)$
```

```
$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{PE^+},\mathrm{Sim}}(1^\lambda)}$:

1: $(X, \mathsf{Msg}, C^*) \leftarrow \mathsf{A}(1^\lambda)$
2: $\mathsf{PE^+.mpk} \leftarrow \mathrm{Sim}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}, C^*)$
3: For $\mathbf{x}_i \in X$, let $b_i = m_i$ if $C^*(\mathbf{x}_i) = 1$,
   $\perp$ otherwise.
4: Let $\mathsf{CT}_X :=$
   $\{\mathsf{PE^+.ct}_{\mathsf{x_i}} \leftarrow \mathrm{Sim}(1^{|\mathbf{x}_i|}, C^*, b_i)\}_{i\in[|X|]}$
5: $\mathsf{PE^+.sk}_{C^*} \leftarrow \mathrm{Sim}(C^*)$
6: $\alpha \leftarrow \mathsf{A}^{\mathrm{Sim}}(\mathsf{CT}_X, \mathsf{PE^+.sk}_{C^*})$
7: Output $(X, \mathsf{Msg}, \alpha)$
```

Here, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|X|}\}$ is the target set of attributes of length inp, and let $\mathsf{Msg} = \{m_1, \ldots, m_{|X|}\}$ be the corresponding set of messages in $\mathcal{M}$. We say an adversary A is admissible if:

1. For a single query C^*, it may hold that $C^*(\mathbf{x}) = 1$ or $C^*(\mathbf{x}) = 0$ for any $\mathbf{x} \in X$.
2. For all other queries $C_i \neq C^*$, it holds that $C_i(\mathbf{x}) = 0$ for any $\mathbf{x} \in X$.

The functional encryption scheme CktFE is then said to be $(1, \mathrm{poly})$-Sel-SIM-secure if there is an admissible stateful PPT simulator Sim such that for every admissible PPT adversary A, the following two distributions are computationally indistinguishable.

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

For the (Q, poly) version of the above game, we replace each occurrence of C^* with a tuple $C_1^*, \ldots, C_Q^*$ and set $b_i = m_i$ if there is $j \in [Q]$ such that $C_j^*(\mathbf{x}_i) = 1$ and otherwise $b_i = \perp$.

Remark B.1.3. Note that the above definition is a multi-challenge one, where the adversary can obtain multiple challenge ciphertexts. While this security notion implies more standard notion of single-challenge security, the latter may not imply the former since the simulator is stateful and may use some secret information to simulate the game. Because of the reason, it is impossible to perform the hybrid argument to prove the implication. However, in the special case of Agrawal's PE^+ scheme [Agrawal (2017)], which is only proven secure in the single-challenge setting, actually satisfies the above stronger notion, since the simulator for the ciphertext does not use any secret information not known to the adversary in her construction.

In our construction of PE^+ for NFA in Appendix B.2, we will use the scheme by [Agrawal (2017)] as a building block. The following theorem summarizes the efficiency properties of her construction.

Theorem B.1.4 (Adapted from [Agrawal (2017)]). *There exists a selectively secure FE scheme* $\mathsf{PE}^+ = (\mathsf{PE}^+.\mathsf{Setup}, \mathsf{PE}^+.\mathsf{KeyGen}, \mathsf{PE}^+.\mathsf{Enc}, \mathsf{PE}^+.\mathsf{Dec})$ *with the following properties under the LWE assumption.*

1. *The circuit* $\mathsf{PE}^+.\mathsf{Setup}(\cdot,\cdot,\cdot;\cdot)$*, which takes as input* $1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}$*, and a randomness* r *and outputs* $\mathsf{PE}^+.\mathsf{msk} = \mathsf{PE}^+.\mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}}; r)$*, can be implemented with depth* $\mathrm{poly}(\lambda, \mathsf{d})$*. In particular, the depth of the circuit is independent of* inp *and the length of the randomness* r*.*
2. *We have* $|\mathsf{PE}^+.\mathsf{sk}_C| \le \mathrm{poly}(\lambda, \mathsf{d})$ *for any* $C \in \mathcal{C}_{\mathsf{inp},\mathsf{d}}$*, where* $(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}) \leftarrow \mathsf{PE}^+.\mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}}, 1^{\mathsf{d}})$ *and* $\mathsf{PE}^+.\mathsf{sk}_C \leftarrow \mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}, C)$*. In particular, the length of the secret key is independent of the input length* inp *and the size of the circuit* C*.*
3. *Let* $C : \{0,1\}^{\mathsf{inp}+\ell} \to \{0,1\}$ *be a circuit such that we have* $C[v] \in \mathcal{C}_{\mathsf{inp},d}$ *for any* $v \in \{0,1\}^\ell$*. Then, the circuit* $\mathsf{PE}^+.\mathsf{KeyGen}(\cdot,\cdot,C[\cdot];\cdot)$*, that takes as input* $\mathsf{PE}^+.\mathsf{mpk}$*,* $\mathsf{PE}^+.\mathsf{msk}$*,* v*, and randomness* r *and outputs* $\mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}, C[v]; r)$*, can be implemented with depth* $\mathsf{depth}(C) \cdot \mathrm{poly}(\lambda, \mathsf{d})$*.*

Proof. The proof follows from the proof of Theorem 3.5.9 and the structure of the PE^+ scheme of [Agrawal (2017)]. We note that the PE^+ scheme of [Agrawal (2017)] uses fully homomorphic encryption (as in [Gorbunov *et al.* (2015)]), to hide the attributes in the ABE scheme of [Boneh *et al.* (2014)], and reverses the FHE encryption by augmenting the circuit in the function key with an FHE decryption circuit. Thus, if a key is requested for a circuit C, then the PE^+ scheme must construct a function key for a circuit $\hat{C} \circ \mathsf{IP}$ where $\hat{C}$ is the FHE evaluation circuit corresponding to circuit C, and IP is the FHE decryption procedure, which in turn, involves computing an inner product followed by a modular reduction [Brakerski *et al.* (2012); Gentry *et al.* (2013)]. Since the FHE evaluation circuit corresponding to some circuit C only causes constant polynomial blowup in depth [Brakerski *et al.* (2012); Gentry *et al.* (2013)] and the FHE decryption circuit is in NC_1, we have that the depth of the augmented circuit $\hat{C} \circ \mathsf{IP}$ is $\mathrm{poly}(\lambda) \cdot \mathsf{depth}(C)$. The setup and encryption algorithms of the PE^+ scheme of [Agrawal (2017)] are the same as that of [Boneh *et al.* (2014)]. Hence, the theorem follows from the proof of Theorem 3.5.9. □

B.1.2 Predicate Encryption and Bounded Key Functional Encryption for NFA

A secret-key functional encryption scheme NfaPE^+ for a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four algorithms. In the following, we fix some alphabet $\Sigma = \Sigma_\lambda$ of size $2 \leq |\Sigma| \leq \mathrm{poly}(\lambda)$.

- $\mathsf{NfaPE}^+.\mathsf{Setup}(1^\lambda)$ is a PPT algorithm takes as input the unary representation of the security parameter and outputs the master secret key $\mathsf{NfaPE}^+.\mathsf{msk}$.
- $\mathsf{NfaPE}^+.\mathsf{Enc}(\mathsf{NfaPE}^+.\mathsf{msk}, \mathbf{x}, m)$ is a PPT algorithm that takes as input the master secret key $\mathsf{NfaPE}^+.\mathsf{msk}$, a string $\mathbf{x} \in \Sigma^*$ of arbitrary length and a message $m \in \mathcal{M}$. It outputs a ciphertext $\mathsf{NfaPE}^+.\mathsf{ct}$.
- $\mathsf{NfaPE}^+.\mathsf{KeyGen}(\mathsf{NfaPE}^+.\mathsf{msk}, M)$ is a PPT algorithm that takes as input the master secret key $\mathsf{NfaPE}^+.\mathsf{msk}$ and a description of an NFA machine M. It outputs a corresponding secret key $\mathsf{NfaPE}^+.\mathsf{sk}_M$.
- $\mathsf{NfaPE}^+.\mathsf{Dec}(\mathsf{NfaPE}^+.\mathsf{sk}_M, M, \mathsf{NfaPE}^+.\mathsf{ct})$ is a deterministic polynomial time algorithm that takes as input the secret key $\mathsf{NfaPE}^+.\mathsf{sk}_M$, its associated NFA M, and a ciphertext $\mathsf{NfaPE}^+.\mathsf{ct}$ and outputs either a message m' or $\perp$.

Remark B.1.5. As in the construction in Sec. 3.6.2, we will pass an additional parameter $\mathsf{s} = \mathsf{s}(\lambda)$ to the $\mathsf{NfaPE}^+.\mathsf{Setup}, \mathsf{NfaPE}^+.\mathsf{Enc}, \mathsf{NfaPE}^+.\mathsf{KeyGen}$ algorithms denoting the description size of NFAs that the scheme can deal with. The construction in Sec. 3.7 can be adapted in a straightforward way to support NFAs with arbitrary size.

Definition B.1.6 (Correctness)**.** A scheme NfaPE^+ is correct if for all NFAs M, all $\mathbf{x} \in \Sigma^*$ such that $M(\mathbf{x}) = 1$ and for all messages $m \in \mathcal{M}$,

$$\Pr\left[\begin{array}{l} \mathsf{NfaPE}^+.\mathsf{msk} \leftarrow \mathsf{NfaPE}^+.\mathsf{Setup}(1^\lambda)\ , \\ \mathsf{NfaPE}^+.\mathsf{sk}_M \leftarrow \mathsf{NfaPE}^+.\mathsf{KeyGen}(\mathsf{NfaPE}^+.\mathsf{msk}, M)\ , \\ \mathsf{NfaPE}^+.\mathsf{ct} \leftarrow \mathsf{NfaPE}^+.\mathsf{Enc}(\mathsf{NfaPE}^+.\mathsf{msk}, \mathbf{x}, m)\ : \\ \mathsf{NfaPE}^+.\mathsf{Dec}\big(\mathsf{NfaPE}^+.\mathsf{sk}_M, M, \mathsf{NfaPE}^+.\mathsf{ct}\big) \neq \mathsf{m} \end{array}\right] = \mathrm{negl}(\lambda)$$

where the probability is taken over the coins of NfaPE^{+}.Setup, NfaPE^{+}.KeyGen, and NfaPE^{+}.Enc.

Definition B.1.7 ($(1, \mathrm{poly})$-Sel-SIM Security)**.** The definition is adapted from Defn B.1.2 in the symmetric key setting. For a stateful PPT adversary A and a stateful PPT simulator Sim, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{NfaPE}^{+},\mathsf{A}}(1^{\lambda})\text{:}}$

```
1: $(X, \mathsf{Msg}, M^*) \leftarrow \mathsf{A}(1^{\lambda})$
2: For $\mathbf{x}_i \in X$, let $b_i = m_i$.
   Let $\mathsf{NfaPE}^{+}.\mathsf{msk} \leftarrow \mathsf{NfaPE}^{+}.\mathsf{Setup}(1^{\lambda})$.
3: Let $\mathsf{CT}_X :=$
   $\{\mathsf{PE}^{+}.\mathsf{ct}_{\mathbf{x}_i} \leftarrow \mathsf{NfaPE}^{+}.\mathsf{Enc}(\ \mathsf{NfaPE}^{+}.\mathsf{msk}, \mathsf{x_i}, \mathsf{b_i}\ )\}_{\mathsf{i}\in[|\mathsf{X}|]}$
4: $\mathsf{PE}^{+}.\mathsf{sk}_{M^*} \leftarrow \mathsf{NfaPE}^{+}.\mathsf{KeyGen}(\mathsf{NfaPE}^{+}.\mathsf{msk}, M^*)$
5: $\alpha \leftarrow \mathsf{A}^{\mathsf{NfaPE}^{+}.\mathsf{KeyGen}(\mathsf{NfaPE}^{+}.\mathsf{msk},\cdot)}(\mathsf{CT}_X, \mathsf{PE}^{+}.\mathsf{sk}_{M^*})$
6: Output $(X, \mathsf{Msg}, \alpha)$
```

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{NfaPE}^{+},\mathrm{Sim}}(1^{\lambda})\text{:}}$

```
1: $(X, \mathsf{Msg}, M^*) \leftarrow \mathsf{A}(1^{\lambda})$
2: For $\mathbf{x}_i \in X$, let $b_i = m_i$ if $M^*(\mathbf{x}_i) = 1$,
   $\perp$ otherwise.
3: let $\mathsf{CT}_X :=$
   $\{\mathsf{PE}^{+}.\mathsf{ct}_{\mathbf{x}_i} \leftarrow \mathrm{Sim}(1^{|\mathsf{x}_i|}, \mathsf{M}^*, \mathsf{b_i})\}_{\mathsf{i}\in[|\mathsf{X}|]}$
4: $\mathsf{PE}^{+}.\mathsf{sk}_{M^*} \leftarrow \mathrm{Sim}(M^*)$
5: $\alpha \leftarrow \mathsf{A}^{\mathrm{Sim}}(\mathsf{CT}_X, \mathsf{PE}^{+}.\mathsf{sk}_{M^*})$
6: Output $(X, \mathsf{Msg}, \alpha)$
```

Here, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|X|}\}$ is the target set of attributes over Σ^*, and let $\mathsf{Msg} = \{m_1, \ldots, m_{|X|}\}$ be the corresponding set of messages in $\mathcal{M}$. We say an adversary A is admissible if:

1. For a single query M^*, it may hold that $M^*(\mathbf{x}_i) = 1$ or $M^*(\mathbf{x}_i) = 0$ for any $\mathbf{x}_i \in X$.
2. For all other queries $M_j \neq M^*$, it holds that $M_j(\mathbf{x}_i) = 0$ for any $\mathbf{x}_i \in X$.

The PE^{+} scheme NfaPE^{+} is then said to be $(1, \mathrm{poly})$-Sel-SIM-secure if there is an admissible stateful PPT simulator Sim such that for every admissible PPT adversary A, the following two distributions are computationally indistinguishable.

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

For the (Q, poly) version of the above game, we replace each occurrence of M^* with a tuple $M_1^*, \ldots, M_Q^*$ and set $b_i = m_i$ if there is $j \in [Q]$ such that $M_j^*(\mathbf{x}_i) = 1$ and otherwise $b_i = \perp$.

B.1.3 Symmetric Key Functional Encryption

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

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

- $\mathsf{SKFE.Setup}(1^\lambda)$ is a PPT algorithm takes as input the unary representation of the security parameter and outputs the master secret key msk.
- $\mathsf{SKFE.KeyGen}(\mathsf{msk}, g)$ is a PPT algorithm that takes as input the master secret key msk and a circuit $g \in \mathcal{F}_\lambda$ and outputs a corresponding secret key sk_g.
- $\mathsf{SKFE.Enc}(\mathsf{msk}, \mathsf{x})$ is a PPT algorithm that takes as input the master secret key msk and an input message $x \in \mathcal{X}_\lambda$ and outputs a ciphertext ct.
- $\mathsf{SKFE.Dec}(\mathsf{sk}_g, \mathsf{ct}_\mathsf{x})$ is a deterministic algorithm that takes as input the secret key sk_g and a ciphertext ct_x and outputs $g(x)$.

Definition B.1.8 (Correctness)**.** A symmetric key functional encryption scheme SKFE is correct if for all $g \in \mathcal{F}_\lambda$ and all $x \in \mathcal{X}_\lambda$,

$$\Pr\left[\begin{array}{l} \mathsf{msk} \leftarrow \mathsf{SKFE.Setup}(1^\lambda); \\ \mathsf{SKFE.Dec}\Big(\mathsf{SKFE.KeyGen}(\mathsf{msk}, g), \mathsf{SKFE.Enc}(\mathsf{msk}, \mathsf{x})\Big) \neq \mathsf{g}(\mathsf{x}) \end{array}\right] = \mathrm{negl}(\lambda)$$

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

Security. In this paper we will consider the standard indistinguishability based definition.

Definition B.1.9. A symmetric key functional encryption scheme SKFE for a function family $\mathcal{F}$ is very selectively secure under the unbounded collusion, if for all PPT adversaries A, the advantage of A in the following experiment is negligible in the security parameter λ:

1. **Key Queries and Challenge Queries.** Given the security parameter 1^λ, A submits key queries $g_1, \ldots, g_q \in \mathcal{F}_\lambda$ and ciphertext queries $(x_1^{(0)}, \ldots, x_{q'}^{(0)}), (x_1^{(1)}, \ldots, x_{q'}^{(1)})$

to the challenger. Here, the number of key queries and the challenge queries can be arbitrarily large polynomial. These queries should satisfy $g_i(x_j^{(0)}) = g_i(x_j^{(1)})$ for all $i \in [q]$ and $j \in [q']$.

2. **Challenge.** Then, the challenger runs $\mathsf{msk} \leftarrow \mathsf{SKFE.Setup}(1^\lambda)$ and chooses random bit b. Then it computes $\mathsf{sk}_i \leftarrow \mathsf{SKFE.KeyGen}(\mathsf{msk}, g_i)$ for $i \in [q]$ and $\mathsf{ct_j} \leftarrow \mathsf{SKFE.Enc}(\mathsf{msk}, \mathsf{x_j^{(b)}})$ for $j \in [q']$ and gives $\{\mathsf{sk}_i\}_{i\in[q]}$ and $\{\mathsf{ct_j}\}_{\mathsf{j}\in[\mathsf{q}']}$ to the adversary.

3. **Guess.** A outputs a bit b', and succeeds if $b' = b$.

The *advantage* of A is $|\Pr[b = b'] - 1/2|$ in the above game.

Function Class. For our purposes, we consider two function classes for SKFE. The first one is circuits, namely we set $\mathcal{X}_\lambda = \{0,1\}^{\mathsf{inp}(\lambda)}$ and $\mathcal{Y}_\lambda$ is the circuits of input length $\mathsf{inp}(\lambda)$ and fixed depth $\mathsf{depth}(\lambda)$ and output length $\mathsf{out}(\lambda)$. The second class is DFAs, namely we set $\mathcal{X}_\lambda = \Sigma^*$ and $\mathcal{Y}_\lambda$ is DFA with alphabet Σ.

B.2 Construction: Predicate and Bounded Key Functional Encryption for NFA

We construct a secret key predicate and bounded key FE scheme for NFA denoted by $\mathsf{NfaPE}^+ = (\mathsf{NfaPE}^+.\mathsf{Setup}, \mathsf{NfaPE}^+.\mathsf{KeyGen}, \mathsf{NfaPE}^+.\mathsf{Enc}, \mathsf{NfaPE}^+.\mathsf{Dec})$ from the following ingredients:

1. $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: a pseudorandom function, where a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ defines a function $\mathsf{PRF.Eval}(\mathsf{K}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$. We denote the length of K by $|\mathsf{K}|$.

2. $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$: a functional encryption scheme for circuit with the efficiency property described in Item 1 of Theorem 3.5.15. We can instantiate FE with the scheme proposed by [Goldwasser *et al.* (2013*a*)].

3. $\mathsf{PE}^+ = (\mathsf{PE}^+.\mathsf{Setup}, \mathsf{PE}^+.\mathsf{KeyGen}, \mathsf{PE}^+.\mathsf{Enc}, \mathsf{PE}^+.\mathsf{Dec})$: A PE^+ scheme for circuits that satisfies the efficiency properties described in Theorem B.1.4. We can instantiate PE^+ with the scheme proposed by [Agrawal (2017)].

4. $U(\cdot,\cdot)$: a universal circuit that takes as input a circuit C of fixed depth and size and an input $\mathbf{x}$ to the circuit and outputs $C(\mathbf{x})$. We often denote by $U[C](\cdot) = U(C, \cdot)$ a universal circuit U with the first input C being hardwired. We need to have $\mathsf{depth}(U) \le O(\mathsf{depth}(C))$. For construction of such a universal circuit, we refer to [Cook and Hoover, (1985)].

Below we provide our construction for secret key PE^+ for NFA, where size of machines is s. This restriction can be removed by the same trick as in Sec. 3.7. In the description below, we abuse notation and denote as if the randomness used in a PPT algorithm was a key K of the pseudorandom function PRF. Namely, for a PPT algorithm (or circuit) A that takes as input x and a randomness $r \in \{0,1\}^\ell$ and outputs y, $\mathsf{A}(x;\mathsf{K})$ denotes an algorithm that computes $r := \mathsf{PRF}(\mathsf{K},1)\|\mathsf{PRF}(\mathsf{K},2)\|\cdots\|\mathsf{PRF}(\mathsf{K},\ell)$ and runs $\mathsf{A}(x;r)$. Note that if A is a circuit, this transformation makes the size of the circuit polynomially larger and adds a fixed polynomial overhead to its depth. In particular, even if we add this change to $\mathsf{PE}^+.\mathsf{Setup}$ and $\mathsf{PE}^+.\mathsf{KeyGen}$, the efficiency properties of PE^+ described in Theorem B.1.4 is preserved.

$\mathsf{NfaPE}^+.\mathsf{Setup}(1^\lambda, 1^\mathsf{s})$: On input the security parameter 1^λ and a description size s of an NFA, do the following:

1. For all $j \in [0,\lambda]$, sample PRF keys $\widehat{\mathsf{K}}_j, \mathsf{R}_j \leftarrow \mathsf{PRF}.\mathsf{Setup}(1^\lambda)$.
2. For all $j \in [0,\lambda]$, sample $(\mathsf{FE}.\mathsf{mpk}_j, \mathsf{FE}.\mathsf{msk}_j) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{d}(\lambda)}, 1^{\mathsf{out}(\lambda)})$.

   Here, we generate $\lambda+1$ instances of FE. Note that all instances support a circuit class with input length $\mathsf{inp}(\lambda) = \mathsf{s} + 2|\mathsf{K}|$, output length $\mathsf{out}(\lambda)$, and depth $\mathsf{d}(\lambda)$, where $\mathsf{out}(\lambda)$ and $\mathsf{d}(\lambda)$ are polynomials in the security parameter that will be specified later.
3. Output $\mathsf{NfaPE}^+.\mathsf{msk} = (\{\widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{FE}.\mathsf{mpk}_j, \mathsf{FE}.\mathsf{msk}_j\}_{j\in[0,\lambda]})$.

$\mathsf{NfaPE}^+.\mathsf{Enc}(\mathsf{NfaPE}^+.\mathsf{msk}, \mathbf{x}, m, 1^\mathsf{s})$: On input the master secret key $\mathsf{NfaPE}^+.\mathsf{msk}$, an attribute $\mathbf{x} \in \Sigma^*$ of length at most 2^λ, a message m and the bound s on NFA size, do the following:

1. Parse the master secret key as $\mathsf{NfaPE}^+.\mathsf{msk} \to (\{\widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{FE}.\mathsf{mpk}_j, \mathsf{FE}.\mathsf{msk}_j\}_{j\in[0,\lambda]})$.
2. Set $\hat{\mathbf{x}} = \mathbf{x}\|\bot^{2^i-\ell}$, where $\ell = |\mathbf{x}|$ and $i = \lceil \log \ell \rceil$.
3. Compute a PE^+ key pair $(\mathsf{PE}^+.\mathsf{mpk}_i, \mathsf{PE}^+.\mathsf{msk}_i) = \mathsf{PE}^+.\mathsf{Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}}_i)$ with $\widehat{\mathsf{K}}_i$ as the randomness.

   Here, we generate an instance of PE^+ that supports a circuit class with input domain $\{0,1\}^{2^i\eta} \supseteq (\Sigma \cup \{\bot\})^{2^i}$ and depth $\hat{\mathsf{d}}$.
4. Compute $\mathsf{PE}^+.\mathsf{ct} \leftarrow \mathsf{PE}^+.\mathsf{Enc}(\mathsf{PE}^+.\mathsf{mpk_i}, \hat{\mathbf{x}}, \mathsf{m})$ as an PE^+ ciphertext for the message m under attribute $\hat{\mathbf{x}}$.
5. Obtain $\mathsf{FE}.\mathsf{sk}_i = \mathsf{FE}.\mathsf{KeyGen}(\mathsf{FE}.\mathsf{mpk}_i, \mathsf{FE}.\mathsf{msk}_i, C_{\mathsf{s},2^i}; \mathsf{R}_i)$, where $C_{\mathsf{s},2^i}$ is a circuit described in Figure B.1.
6. Output $\mathsf{NfaPE}^+.\mathsf{ct} = (\mathsf{FE}.\mathsf{sk_i}, \mathsf{PE}^+.\mathsf{mpk_i}, \mathsf{PE}^+.\mathsf{ct})$.

Without loss of generality, we assume that i is revealed from $\mathsf{PE}^+.\mathsf{ct}$.

Function $C_{\mathsf{s},2^i}$

(a) Parse the input $\mathbf{w} = (M, \widehat{\mathsf{K}}, \widehat{\mathsf{R}})$, where M is an NFA and $\widehat{\mathsf{K}}$ and $\widehat{\mathsf{R}}$ are PRF keys.

(b) Compute $(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}) = \mathsf{PE}^+.\mathsf{Setup}(1^\lambda, 1^{2^i \eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}})$.

(c) Compute $\widehat{M_{2^i}} = \mathsf{To\text{-}Circuit}_{\mathsf{s},2^i}(M)$. (See Theorem 3.6.1 for the definition of $\mathsf{To\text{-}Circuit}$.)

(d) Compute and output $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M_{2^i}}]} = \mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}, \mathsf{PE}^+.\mathsf{msk}, U[\widehat{M_{2^i}}]; \widehat{\mathsf{R}})$.

Figure B.1: Circuit $C_{\mathsf{s},2^i}$, supported by the FE scheme. $C_{\mathsf{s},2^i}$ takes NFA M as input and outputs a secret key for the universal circuit $U[\widehat{M_{2^i}}]$ (hardwired with $\widehat{M_{2^i}}$) under the PE^+ scheme.

$\mathsf{NfaPE}^+.\mathsf{KeyGen}(\mathsf{NfaPE}^+.\mathsf{msk}, M, 1^{\mathsf{s}})$: On input the master secret key $\mathsf{NfaPE}^+.\mathsf{msk}$, the description of an NFA M and a size s of the NFA, if $|M| \neq \mathsf{s}$, output $\perp$ and abort. Else, proceed as follows.

1. Parse the master secret key as $\mathsf{NfaPE}^+.\mathsf{msk} \to (\{\widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{FE}.\mathsf{mpk}_j, \mathsf{FE}.\mathsf{msk}_j\}_{j\in[0,\lambda]})$.
2. Sample $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF}.\mathsf{Setup}(1^\lambda)$ for all $j \in [0, \lambda]$.
3. Compute $\mathsf{FE}.\mathsf{ct}_j \leftarrow \mathsf{FE}.\mathsf{Enc}(\mathsf{FE}.\mathsf{mpk}_j, (M, \widehat{\mathsf{K}}_j, \widehat{\mathsf{R}}_j))$ for all $j \in [0, \lambda]$.
4. Output $\mathsf{NfaPE}^+.\mathsf{sk}_M = \{\mathsf{FE}.\mathsf{ct}_j\}_{j\in[0,\lambda]}$.

$\mathsf{NfaPE}^+.\mathsf{Dec}(\mathsf{NfaPE}^+.\mathsf{sk}_M, M, \mathsf{NfaPE}^+.\mathsf{ct})$: On input a secret key for NFA M and a ciphertext, proceed as follows:

1. Parse the secret key as $\mathsf{NfaPE}^+.\mathsf{sk}_M \to \{\mathsf{FE}.\mathsf{ct}_j\}_{j\in[0,\lambda]}$ and the ciphertext as $\mathsf{NfaPE}^+.\mathsf{ct} \to (\mathsf{FE}.\mathsf{sk}_\mathsf{i}, \mathsf{PE}^+.\mathsf{mpk}_\mathsf{i}, \mathsf{PE}^+.\mathsf{ct})$.
2. Learn i from $\mathsf{PE}^+.\mathsf{ct}$ and choose $\mathsf{FE}.\mathsf{ct}_i$ from $\mathsf{NfaPE}^+.\mathsf{sk}_M = \{\mathsf{FE}.\mathsf{ct}_j\}_{j\in[0,\lambda]}$.
3. Compute $y = \mathsf{FE}.\mathsf{Dec}(\mathsf{FE}.\mathsf{mpk}_i, \mathsf{FE}.\mathsf{sk}_i, \mathsf{FE}.\mathsf{ct}_i)$.
4. Compute and output $z = \mathsf{PE}^+.\mathsf{Dec}(\mathsf{PE}^+.\mathsf{mpk}_i, y, U[\widehat{M_{2^i}}], \mathsf{PE}^+.\mathsf{ct}_\mathsf{i})$, where we interpret y as a secret key.

Correctness follows as in Section 3.6.3 by appropriately setting $\mathsf{out}(\lambda)$ and $\mathsf{d}(\lambda)$. For security, we prove the following theorem.

Theorem B.2.1. *Assume that* FE *satisfies full simulation based security,* PE^+ *is* $\mathsf{Sel\text{-}SIM}$ *secure, and that* PRF *is a secure pseudorandom function. Then,* NfaPE^+ *satisfies selective simulation based security.*

We note that the structure of hybrids is almost the same as in the proof of Theorem 3.6.4. The difference is that instead of changing ciphertexts corresponding to each instance of PE^+ from an encryption of m_0 to m_1, we change honest setup, key generation, and encryption of each instance of PE^+ to the simulated ones. To do so, we must replace the reliance on ABE for circuits with PE^+ for circuits. In more detail, we consider $\mathbf{Game}_i$ for $i \in [0, i_{\max}+1]$ as follows, where $i_{\max}$ is defined as in the proof of Theorem 3.6.4.

$\mathbf{Game}_i$**:** The game proceeds as follows. In the following $\mathsf{FE}.\mathrm{Sim}$ and $\mathsf{PE}^+.\mathrm{Sim}$ are the simulators for FE and PE^+, respectively.

Setup phase. At the beginning of the game, A takes 1^λ as input and submits 1^{s} and (X, Msg, M^*) to the challenger. Then, the challenger chooses $\{\widehat{\mathsf{K}}_j, \mathsf{R}_j\}_{j\in[0,\lambda]}$ and $\{\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j\}_{j\in[0,\lambda]}$. It further samples $\mathsf{PE}^+.\mathsf{mpk}_j \leftarrow \mathsf{PE}^+.\mathrm{Sim}(1^\lambda, 1^{2^j\eta}, 1^{\hat{\mathsf{d}}}, U[\widehat{M^*_{2^j}}])$ and $\mathsf{FE.sk}_j \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j, C_{\mathsf{s},2^j})$ for $j \leq i-1$.

The challenger answers the encryption and key queries made by A as follows.

Simulating Keys. During the game, secret key for M (which is possibly M^*) is answered as follows. For $j \in [0, \lambda]$, the challenger computes

$$\mathsf{FE.ct}_j \leftarrow \begin{cases} \mathsf{FE.Enc}(\mathsf{FE.mpk}_j, (M, \widehat{\mathsf{K}}_j, \widehat{\mathsf{R}}_j)) & \text{If } \lambda \geq j \geq i \\ \mathsf{FE}.\mathrm{Sim}(\mathsf{FE.mpk}_j, \mathsf{FE.sk}_j, C_{\mathsf{s},2^j}, \mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M}_{2^j}]}, 1^{\mathsf{inp}(\lambda)}) & \text{If } j \leq i-1, \end{cases} \tag{B.1}$$

where $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M}_{2^j}]} \leftarrow \mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}_j, \mathsf{PE}^+.\mathsf{msk}_j, U[\widehat{M_{2^j}}])$ and returns $\{\mathsf{FE.ct}_j\}$ to A.

Simulating Ciphertexts. To generate a ciphertext for $m \in \mathsf{Msg}$ associated with $\mathbf{x} \in X$, the challenger sets $j := \lceil \log |\mathbf{x}| \rceil$ and computes

$$\mathsf{PE}^+.\mathsf{ct} \leftarrow \begin{cases} \mathsf{PE}^+.\mathsf{Enc}(\mathsf{PE}^+.\mathsf{mpk_j}, \hat{\mathbf{x}}, \mathsf{m}) & \text{If } i_{\max} \geq j \geq i \\ \mathsf{PE}^+.\mathrm{Sim}(1^{2^j\eta}, \widehat{M^*_{2^j}}, b) & \text{If } j \leq i-1, \end{cases}$$

where $b = m$ if $M^*(\mathbf{x}) = 1$ and $\perp$ otherwise. It also computes $\mathsf{FE.sk}_j = \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j, C_{\mathsf{s},2^j}; \mathsf{R}_j)$ if $j \geq i$. The ciphertext is $\mathsf{NfaPE}^+.\mathsf{ct} = (\mathsf{FE.sk_j}, \mathsf{PE}^+.\mathsf{mpk_j}, \mathsf{PE}^+.\mathsf{ct})$.

Finally, A outputs its guess b'.

It is easy to see that $\mathbf{Game}_0$ is the same as $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{NfaPE}^+,\mathsf{A}}(1^\lambda)$. Furthermore, we can construct a simulator for NfaPE^+ from the challenger in $\mathbf{Game}_{i_{\max}+1}$ appropriately, since the challenger in $\mathbf{Game}_{i_{\max}+1}$ only uses b and $|\mathbf{x}|$ to simulate the ciphertext. Therefore, it suffices to show the indistinguishability between $\mathbf{Game}_i$ and $\mathbf{Game}_{i+1}$. To do so, we further consider following sequence of games, which closely follows the proof of Theorem 3.6.4 except that we do not need counterpart of $\mathbf{Game}_{i,6}$, where we undo changes we made from $\mathbf{Game}_{i,1}$ to $\mathbf{Game}_{i,5}$. We directly go to $\mathbf{Game}_{i+1,0}$ from $\mathbf{Game}_{i,5}$.

$\mathbf{Game}_{i,0}$**:** The game is the same as $\mathbf{Game}_i$.

$\mathbf{Game}_{i,1}$**:** The game is the same as the previous game except that $(\mathsf{PE}^+.\mathsf{mpk}_i, \mathsf{PE}^+.\mathsf{msk}_i)$ and $\mathsf{FE}.\mathsf{sk}_i$ are computed at the setup phase using $\widehat{\mathsf{K}}_i$ and R_i.

$\mathbf{Game}_{i,2}$**:** The game is the same as the previous game except that $\mathsf{FE}.\mathsf{sk}_i$ is generated using true randomness instead of using the PRF keys.

$\mathbf{Game}_{i,3}$**:** In this game, to answer a key query, $\mathsf{FE}.\mathsf{ct_i}$ is computed using $\mathsf{FE}.\mathrm{Sim}$ on input $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M}_{2^i}]} = \mathsf{PE}^+.\mathsf{KeyGen}(\mathsf{PE}^+.\mathsf{mpk}_i, \mathsf{PE}^+.\mathsf{msk}_i, U[\widehat{M}_{2^i}]; \widehat{\mathsf{R}}_i)$.

$\mathbf{Game}_{i,4}$**:** In this game, to answer a key query, $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M}_{2^i}]}$ is generated using true randomness instead of using the PRF key.

$\mathbf{Game}_{i,5}$**:** In this game, we generate $\mathsf{PE}^+.\mathsf{mpk}_i$ using $\mathsf{PE}^+.\mathrm{Sim}$. We also generate $\mathsf{PE}^+.\mathsf{ct}$ using $\mathsf{PE}^+.\mathrm{Sim}$ when $\lceil \log |\mathbf{x}| \rceil = i$.

Note that we have $\mathbf{Game}_{i,5} = \mathbf{Game}_{i+1,0}$. Furthermore, indistinguishability between $\mathbf{Game}_{i,j-1}$ and $\mathbf{Game}_{i,j}$ for $j \in [4]$ can be established in exactly the same manners as in Theorem 3.6.4. For $\mathbf{Game}_{i,4}$ and $\mathbf{Game}_{i,5}$, we give the proof in the following.

Lemma B.2.2. *We have* $|\Pr[\mathsf{E}_{i,4}] - \Pr[\mathsf{E}_{i,5}]| = \mathrm{negl}(\lambda)$.

Proof. To prove the lemma, let us assume that $|\Pr[\mathsf{E}_{i,4}] - \Pr[\mathsf{E}_{i,5}]|$ is non-negligible and construct an adversary B that breaks the selective simulation security of PE^+ using A. B proceeds as follows.

Setup phase. At the beginning of the game, B inputs 1^λ to A and gets (X, Msg, M^*) and 1^{s} from A. Then PE^+ challenger chooses $\mathsf{PE}^+.\mathsf{mpk}$ and sends it to B, where

$\mathsf{PE}^+.\mathsf{mpk}$ is honestly chosen or simulated, depending on whether B is playing the real game or simulated game. B then sets $\mathsf{PE}^+.\mathsf{mpk}_i := \mathsf{PE}^+.\mathsf{mpk}$ and chooses $\widehat{\mathsf{K}}_j, \mathsf{R}_j \leftarrow \mathsf{PRF}.\mathsf{Setup}(1^\lambda)$ and $(\mathsf{FE}.\mathsf{mpk}_j, \mathsf{FE}.\mathsf{msk}_j) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{out}(\lambda)}, 1^{\mathsf{d}(\lambda)})$ for $j \in [0, \lambda]$. It also computes $\mathsf{FE}.\mathsf{sk}_i \leftarrow \mathsf{FE}.\mathsf{KeyGen}(\mathsf{FE}.\mathsf{mpk}_i, \mathsf{FE}.\mathsf{msk}_i, C_{\mathsf{s},2^i})$. Finally, B sets

$$X_i := \{\hat{\mathbf{x}} = \mathbf{x} \| \bot^{2^i - |\mathbf{x}|} : \mathbf{x} \in X, 2^{i-1} < |\mathbf{x}| \leq 2^i\}$$

and

$$\mathsf{Msg}_i := \{m_j \in \mathsf{Msg} : 2^{i-1} < |\mathbf{x}_j| \leq 2^i\}$$

and submits its target as $(X_i, \mathsf{Msg}_i, \widehat{M^*_{2^i}})$ and $(1^{2^i\eta}, 1^{\hat{\mathsf{d}}}, 1^{\mathsf{s}})$ to the PE^+ challenger, where $\widehat{M^*_{2^i}} = \mathsf{To\text{-}Circuit}_{|M^*|,2^i}(M^*)$.

After B specifies its target, it is given a secret key $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M^*_{2^i}}]}$ and ciphertexts $\{\mathsf{PE}^+.\mathsf{ct}_{\hat{\mathbf{x}}}\}_{\hat{\mathbf{x}} \in X_i}$. They are honestly generated or simulated depending on whether B is playing the real game or simulated game. B then simulates the ciphertexts $\{\mathsf{NfaPE}^+.\mathsf{ct}_{\mathbf{x}}\}_{\mathsf{x} \in \mathsf{X}}$ as follows.

Simulating Ciphertexts. To generate a ciphertext for $\mathbf{x} \in X$ associated with $m \in \mathsf{Msg}$, B computes the ciphertext $\mathsf{NfaPE}^+.\mathsf{ct}_{\mathbf{x}}$ as follows. For the case of $\lceil \log |\mathbf{x}| \rceil \neq i$, it proceeds as in the previous game. Otherwise, it retrieves corresponding component $\mathsf{PE}^+.\mathsf{ct}_{\hat{\mathbf{x}}}$ to $\hat{\mathbf{x}}$ from $\{\mathsf{PE}^+.\mathsf{ct}_{\hat{\mathbf{x}}}\}_{\hat{\mathbf{x}} \in X_i}$ and sets $\mathsf{NfaPE}^+.\mathsf{ct}_{\mathbf{x}} = (\mathsf{FE}.\mathsf{sk}_\mathsf{i}, \mathsf{PE}^+.\mathsf{mpk}_\mathsf{i}, \mathsf{PE}^+.\mathsf{ct}_{\hat{\mathbf{x}}})$, where B uses $\mathsf{PE}^+.\mathsf{mpk}_i$ and $\mathsf{FE}.\mathsf{sk}_i$ that are sampled in the setup phase.

B also simulates the 1-key $\mathsf{NfaPE}^+.\mathsf{sk}_{M^*} := \{\mathsf{FE}.\mathsf{ct}_\mathsf{j}\}_{\mathsf{j} \in [0,\lambda]}$ from $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M^*_{2^i}}]}$ as in Eq. (B.1). B then gives $\mathsf{NfaPE}^+.\mathsf{sk}_{M^*}$ and $\{\mathsf{NfaPE}^+.\mathsf{ct}_{\mathbf{x}}\}_{\mathsf{x} \in \mathsf{X}}$ to A and answers the key queries as follows.

Key queries. Given an NFA M of size s from A, B first chooses $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF}.\mathsf{Setup}(1^\lambda)$ for $j \in [0, \lambda]$. It then queries a secret key for $U[\widehat{M_{2^i}}]$ to its challenger. Then, the challenger returns $\mathsf{PE}^+.\mathsf{sk}_{U[\widehat{M_{2^i}}]}$ to B. The key is honestly generated or simulated depending on whether B is playing the real game or simulated game. It then computes $\mathsf{FE}.\mathsf{ct}_\mathsf{j}$ for $j \in [0, \lambda]$ as in Eq. (B.1) and returns $\mathsf{NfaPE}^+.\mathsf{sk}_M := \{\mathsf{FE}.\mathsf{ct}_\mathsf{j}\}_{\mathsf{j} \in [0,\lambda]}$ to A.

It is easy to see that B simulates $\mathbf{Game}_{i,4}$ if it is in the real game and $\mathbf{Game}_{i,5}$ if it is in the simulated game. Therefore, B breaks the security of PE^+ if A distinguishes the two games. It remains to prove that B is a legitimate adversary (i.e., it does not make any prohibited key queries). For any attribute $\hat{\mathbf{x}}$ for which B makes an encryption query and for any circuit $U[\widehat{M}_{2^i}]$ for which B makes a key query, we have

$$U[\widehat{M}_{2^i}](\hat{\mathbf{x}}) = \widehat{M}_{2^i}(\hat{\mathbf{x}}) = M(\mathbf{x}),$$

where the second equality above follows from Item 1 of Theorem 3.6.1. Therefore, B is a legitimate adversary as long as so is A. This completes the proof of the lemma. □

B.3 Definitions: Reusable Garbled Nondeterministic Finite Automata

In this section, we will define garbled NFAs and notions of input and function privacy, adapting corresponding definitions from [Agrawal and Singh (2017)]. We further show how to construct garbled NFAs (with unbounded inputs) that can be used to evaluate multiple inputs (of possibly varying size).

Definition B.3.1. (Garbled NFA scheme) A garbling scheme for a family of NFAs $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ with M_λ a family of NFAs $\Sigma_\lambda \times \mathcal{Q}_\lambda \to \mathcal{Q}_\lambda$, is a tuple of PPT algorithms $\mathsf{RGbNFA} = (\mathsf{RGNfa.Garble}, \mathsf{RGNfa.Encode}, \mathsf{RGNfa.Eval})$ such that

- $\mathsf{RGbNFA.Setup}(1^\lambda)$ takes as input the security parameter λ and outputs a secret key gsk.
- $\mathsf{RGNfa.Garble}(\mathsf{gsk}, M)$ takes as input a secret key gsk and an NFA $M \in \mathcal{M}_\lambda$ and outputs the garbled NFA M_G.
- $\mathsf{RGNfa.Encode}(\mathsf{gsk}, \mathbf{w})$ takes as input the vector $\mathbf{w} \in \Sigma^*$, the secret key gsk and outputs an encoding c.
- $\mathsf{RGNfa.Eval}(M_G, c)$ takes as input a garbled NFA M_G, an encoding c and outputs 1 iff M accepts $\mathbf{w}$, 0 otherwise.

Definition B.3.2. (Correctness). For all sufficiently large security parameters λ, for all NFAs $M \in \mathcal{M}_\lambda$ and all $\mathbf{w} \in \Sigma^*$, we have:

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

Definition B.3.3. (Efficiency) There exist universal polynomials $p_1 = p_1(\lambda)$ and $p_2 = p_2(\lambda, \cdot)$ such that for all security parameters λ, for all NFAs $M \in M_\lambda$, for all $\mathbf{w} \in \Sigma^*$,

$$\Pr\left[\begin{array}{l}\mathsf{gsk} \leftarrow \mathsf{RGbNFA.Setup}(1^\lambda) : \\ |\mathsf{gsk}| \leq p_1(\lambda) \text{ and runtime } (\mathsf{RGNfa.Encode}(\mathsf{gsk}, \mathbf{w})) \leq p_2(\lambda, |\mathbf{w}|)\end{array}\right] = 1.$$

Definition B.3.4. (Input and machine privacy with reusability) Let RGbNFA be a garbling scheme for a family of NFAs $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$. For a pair of stateful PPT algorithms A and a PPT simulator Sim, consider the following two experiments:

```
Exp^real_{RGbNFA,A}(1^λ):
1: $M \leftarrow \mathsf{A}(1^\lambda)$
2: $\mathsf{gsk} \leftarrow \mathsf{RGbNFA.Setup}(1^\lambda)$
   $M_G \leftarrow \mathsf{RGNfa.Garble}(\mathsf{gsk}, M)$
3: $\alpha \leftarrow \mathsf{A}^{\mathsf{RGNfa.Encode}(\mathsf{gsk},\cdot)}(M, M_G)$
4: Output $\alpha$

Exp^ideal_{RGbNFA,A,Sim}(1^λ):
1: $M \leftarrow \mathsf{A}(1^\lambda)$
2: $\tilde{M}_G \leftarrow \mathrm{Sim}(1^\lambda, 1^{|M|})$
3: $\alpha \leftarrow \mathsf{A}^{\mathsf{O}(\cdot,M)}(M, \tilde{M}_G)$
4: Output $\alpha$
```

Here, $\mathsf{O}(\cdot, M)$ is an oracle that on input $\mathbf{w}$ from A, runs Sim with inputs $M(\mathbf{w})$, $1^{|\mathbf{w}|}$, and the latest state of Sim; it returns the output of Sim (Note that Sim updates and maintains its internal states upon its invocation, since it is a stateful algorithm.).

A garbling scheme RGbNFA is input and machine private with reusability if there exists a PPT simulator Sim such that for all pairs of PPT adversaries A, the following two distributions are computationally indistinguishable:

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

Selective Simulation Security. We can consider a weaker version of the above security notion where $\mathcal{A}$ outputs a set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|X|}\} \subset \Sigma^*$ along with M at the beginning of the game and $\mathcal{A}$ is only allowed to query $\mathbf{x} \in X$ to $\mathsf{RGNfa.Encode}(\mathsf{gsk}, \cdot)$ and $\mathsf{O}(\cdot, M)$. We call this security notion selective simulation security.

B.4 Construction: Reusable Garbled NFA

We first note that one could consider replacing the underlying PE^+ scheme in Section B.2 with a Reusable Garbled circuit scheme, RGC to obtain a construction of Reusable Garbled NFA. Following the previous template of our constructions from Sec. 3.6.2 and Sec. B.2, this makes the circuit supported by the underlying FE scheme to work as follows.

1. Convert the input NFA machine M to an equivalent circuit $\widehat{M_{2^i}}$ that can handle inputs of length 2^i.

2. Compute an RGC secret key to encode $\widehat{M_{2^i}}$ and output this as its garbled version.

This intuitive conversion fails to work since the resulting encoding as the garbled machine cannot be shorter than the circuit description length $|\widehat{M_{2^i}}|$, while we need it to be independent of equivalent circuit size. At a high level, to avoid this problem and still ensure machine privacy, we will encrypt M under a symmetric key encryption scheme, SKE. Further, we will use another FE scheme (detailed below) with suitable efficiency guarantees to decrypt this SKE ciphertext encoding M and then run its equivalent circuit description on the encoded input to obtain the desired output.

To this end, we use the following ingredients:

1. $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: a pseudorandom function, where a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ defines a function $\mathsf{PRF.Eval}(\mathsf{K}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$. We denote the length of K by $|\mathsf{K}|$.

2. $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$: a secret key encryption, where the length of a secret key $\mathsf{S} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ is denoted by $|\mathsf{S}|$. For the sake of concreteness and simplicity, we assume that the ciphertext length encrypting a message of length s is $\mathsf{s} + \lambda$. Furthermore, we assume that the depth of the decryption circuit is bounded by some fixed polynomial $\mathrm{poly}(\lambda)$ that is independent of the length of the message. These properties can be easily achieved by using PRF for instance.

3. $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$: a functional encryption scheme for circuit with the efficiency property described in Item 1 of Theorem 3.5.15. We can instantiate FE by the scheme proposed by [Goldwasser *et al.* (2013*a*)].

4. $\widehat{\mathsf{FE}} = (\widehat{\mathsf{FE}}.\mathsf{Setup}, \widehat{\mathsf{FE}}.\mathsf{KeyGen}, \widehat{\mathsf{FE}}.\mathsf{Enc}, \widehat{\mathsf{FE}}.\mathsf{Dec})$: a functional encryption scheme for circuit with the efficiency properties described in Theorem B.1.4. We can

instantiate $\widehat{\mathsf{FE}}$ with the PE^+ scheme proposed by [Agrawal (2017)].[1] Alternatively, we can instantiate $\widehat{\mathsf{FE}}$ by the scheme proposed by Goldwasser et. al with the underlying ABE scheme used in their construction being instantiated by the scheme by [Boneh *et al.* (2014)]. In both cases, the scheme will only have selective simulation security rather than full simulation security. See Remark 3.5.14 for the definition of selective simulation security and Remark 3.5.10 for the reason why we only have *selective* simulation secure FE scheme with the required efficiency properties.

The reusable garbled NFA scheme is defined as follows. The encode algorithm here needs the size s of NFA. This requirement will be removed later using similar technique to Sec. 3.7.

$\mathsf{RGbNFA.Setup}(1^\lambda)$: Upon input the security parameter, sample PRF keys $\mathsf{K}_j, \widehat{\mathsf{K}}_j, \mathsf{R}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ and SKE key $\mathsf{S}_j \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for all $j \in [0, \lambda]$. Then, output $\mathsf{gsk} = (\{\widehat{\mathsf{K}}_j, \mathsf{K}_j, \mathsf{R}_j, \mathsf{S}_j\}_{j\in[0,\lambda]})$.

$\mathsf{RGNfa.Garble}(\mathsf{gsk}, M)$: Upon input a secret key gsk and an NFA machine M of size $|M| = \mathsf{s}$, do the following:

1. For all $j \in [0, \lambda]$, sample $(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{d}(\lambda)}, 1^{\mathsf{out}(\lambda)}; \mathsf{K}_j)$.

   Here, we generate $\lambda + 1$ instances of FE. Note that all instances support a circuit class with input length $\mathsf{inp}(\lambda) = \mathsf{s}+\lambda+2|\mathsf{K}|$, output length $\mathsf{out}(\lambda)$, and depth $\mathsf{d}(\lambda)$, where $\mathsf{out}(\lambda)$ and $\mathsf{d}(\lambda)$ are polynomials in the security parameter as in Section 3.6.
2. Sample $\widehat{\mathsf{R}}_j \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ for all $j \in [0, \lambda]$.
3. Compute $\mathsf{SKE.ct_j} \leftarrow \mathsf{SKE.Enc(S_j, M)}$ for all $j \in [0, \lambda]$.
4. Compute $\mathsf{FE.ct}_j \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_j, (\mathsf{SKE.ct_j}, \widehat{\mathsf{K}}_\mathsf{j}, \widehat{\mathsf{R}}_\mathsf{j}))$ for all $j \in [0, \lambda]$.
5. Output garbled NFA $M_G = (\{\mathsf{SKE.ct_j}, \mathsf{FE.mpk_j}, \mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]})$.

$\mathsf{RGNfa.Encode}(\mathsf{gsk}, \mathbf{x}, 1^\mathsf{s})$: Upon input a secret key gsk, a vector $\mathbf{x}$, and the size of NFA 1^s do the following:

1. Parse gsk as $\mathsf{gsk} \to (\{\widehat{\mathsf{K}}_j, \mathsf{K}_j, \mathsf{R}_j, \mathsf{S}_j\}_{j\in[0,\lambda]})$.
2. Set $\hat{\mathbf{x}} = \mathbf{x}\|\bot^{2^i-\ell}$, where $\ell = |\mathbf{x}|$ and $i = \lceil \log \ell \rceil$.
3. Sample $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{d}(\lambda)}, 1^{\mathsf{out}(\lambda)}; \mathsf{K}_i)$.

[1]Though PE^+ has different syntax from functional encryption, it is easy to convert the former into the latter. For example, we encrypt a random message under an attribute $\mathbf{x}$ using PE^+ and append the message to the PE^+ ciphertext to form a functional encryption of a message $\mathbf{x}$. To decrypt the ciphertext, we use PE^+ secret key to decrypt the PE^+ ciphertext and output 1 if it corresponds to the appended message and 0 otherwise.

4. Sample $(\widehat{\mathsf{FE}}.\mathsf{mpk}_i, \widehat{\mathsf{FE}}.\mathsf{msk}_i) = \widehat{\mathsf{FE}}.\mathsf{Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}(\lambda)}, 1^1; \widehat{\mathsf{K}}_i)$. Note that this FE supports a circuit class with input domain $\{0,1\}^{2^i\eta} \supseteq (\Sigma \cup \{\perp\})^{2^i}$, single bit output, and depth $\hat{\mathsf{d}}$.
5. Compute $\widehat{\mathsf{FE}}.\mathsf{ct} \leftarrow \widehat{\mathsf{FE}}.\mathsf{Enc}(\widehat{\mathsf{FE}}.\mathsf{mpk_i}, (\mathsf{S_i}, \hat{\mathbf{x}}))$.
6. Obtain $\mathsf{FE.sk}_i = \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i, C_{\mathsf{s},2^i}; \mathsf{R}_i)$, where $C_{\mathsf{s},2^i}$ is a circuit described in Figure B.2.

Function $C_{\mathsf{s},2^i}$

(a) Parse the input $\mathbf{w} = (\mathsf{SKE.ct}, \widehat{\mathsf{K}}, \widehat{\mathsf{R}})$, where SKE.ct is an SKE ciphertext and $\widehat{\mathsf{K}}$ and $\widehat{\mathsf{R}}$ are PRF keys.

(b) Compute $(\widehat{\mathsf{FE}}.\mathsf{mpk}, \widehat{\mathsf{FE}}.\mathsf{msk}) = \widehat{\mathsf{FE}}.\mathsf{Setup}(1^\lambda, 1^{2^i\eta}, 1^{\hat{\mathsf{d}}}; \widehat{\mathsf{K}})$

(c) Compute and output $\widehat{\mathsf{FE}}.\mathsf{sk} = \widehat{\mathsf{FE}}.\mathsf{KeyGen}(\widehat{\mathsf{FE}}.\mathsf{mpk}, \widehat{\mathsf{FE}}.\mathsf{msk}, D_{\mathsf{s},2^i}[\mathsf{SKE.ct}]; \widehat{\mathsf{R}})$. (See Figure B.3 for the definition of $D_{\mathsf{s},2^i}[\mathsf{SKE.ct}]$)

Figure B.2: Circuit $C_{\mathsf{s},2^i}$, supported by the FE scheme. $C_{\mathsf{s},2^i}$ takes SKE.ct (encoding NFA M) as input and outputs a secret key under the $\widehat{\mathsf{FE}}$ scheme for another circuit $D_{\mathsf{s},2^i}$ that is also hardwired with SKE.ct.

Function $D_{\mathsf{s},2^i}[\mathsf{SKE.ct}]$

(a) Parse the input $\mathbf{w} = (\mathsf{S}, \hat{\mathbf{x}})$, where S is an SKE secret key.

(b) Compute $M = \mathsf{SKE.Dec}(\mathsf{S}, \mathsf{SKE.ct})$.

(c) Compute $\widehat{M}_{2^i} = \mathsf{To\text{-}Circuit}_{\mathsf{s},2^i}(M)$. (See Theorem 3.6.1 for the definition of To-Circuit.)

(d) Compute and output $\widehat{M}_{2^i}(\hat{\mathbf{x}})$. (This part of the computation is implemented by $U[\widehat{M}_{2^i}]$ where the universal circuit U is instantiated by [Cook and Hoover, (1985)].)

Figure B.3: Circuit $D_{\mathsf{s},2^i}[\mathsf{SKE.ct}]$, supported by the $\widehat{\mathsf{FE}}$ scheme. $D_{\mathsf{s},2^i}[\mathsf{SKE.ct}]$ takes a secret key S (encoding NFA M) and $\widehat{\mathbf{x}}$ as input. It computes and outputs $M(\mathbf{x})$ after decrypting M using S and SKE.ct.

7. Output $c = (\widehat{\mathsf{FE}}.\mathsf{mpk}_i, \mathsf{FE.sk}_i, \widehat{\mathsf{FE}}.\mathsf{ct})$.

$\mathsf{RGNfa.Eval}(M_G, c)$: Upon input the reusable garbled NFA M_G and the input encoding c, do the following:

1. Parse the garbled machine as $M_G \to (\{\mathsf{SKE.ct_j}, \mathsf{FE.mpk_j}, \mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]})$ and the encoding as $c \to (\widehat{\mathsf{FE}}.\mathsf{mpk}_i, \mathsf{FE.sk}_i, \widehat{\mathsf{FE}}.\mathsf{ct})$.
2. Set $\ell = |\mathbf{x}|$ and choose $\mathsf{FE.ct}_i$ such that $i = \lceil \log \ell \rceil < \lambda$.
3. Compute $y = \mathsf{FE.Dec}(\mathsf{FE.mpk}_i, \mathsf{FE.sk}_i, C_{\mathsf{s},2^i}, \mathsf{FE.ct}_i)$.
4. Construct $D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}]$ from $\mathsf{SKE.ct_i}$.
5. Compute and output $z = \mathsf{FE.Dec}(\widehat{\mathsf{FE}}.\mathsf{mpk}_i, y, D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}], \widehat{\mathsf{FE}}.\mathsf{ct})$, where we interpret y as a secret key of the underlying FE.

Correctness. Correctness of the scheme follows from the correctness of FE and $\widehat{\mathsf{FE}}$ as in Section 3.6.4 if we appropriately set $\hat{\mathsf{d}}$, out, and d. Here, we give a brief explanation. We first observe that $D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}]$ can be implemented by combining the decryption circuit of SKE, $\mathsf{To\text{-}Circuit}_{\mathsf{s},2^i}$, and $U[\widehat{M}_{2^i}]$. The depth of the first circuit (resp., the latter two circuits) can be bounded by some fixed polynomial, which is in particular independent of 2^i, by our assumption on SKE (resp., by Theorem 3.6.1). Therefore, the depth of the overall circuit $D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}]$ can be bounded by some fixed polynomial. We would set $\hat{\mathsf{d}}(\lambda)$ to be larger than this polynomial so that we can invoke the correctness of $\widehat{\mathsf{FE}}$. By our assumption on the secret key size of $\widehat{\mathsf{FE}}$, we can bound the length of $\widehat{\mathsf{FE}}.\mathsf{sk}$ that is output by $C_{\mathsf{s},2^i}$ by a polynomial in $\hat{\mathsf{d}}$ and λ, which can be bounded by some fixed polynomial in λ. We would set $\mathsf{out}(\lambda)$ to be larger than this polynomial. Furthermore, we can see that the depth of $C_{\mathsf{s},2^i}$ can be bounded by some fixed polynomial by the assumptions we posed on the depth of the setup and key generation circuits of $\widehat{\mathsf{FE}}$ and by the fact that the depth of $D_{\mathsf{s},2^i}[\mathsf{SKE.ct}]$ can be bounded by some fixed polynomial. We would set $\mathsf{d}(\lambda)$ to be larger than this polynomial. Since the depth and output length of $C_{\mathsf{s},2^i}$ are bounded by d and out respectively, the circuit $C_{\mathsf{s},2^i}$ is supported by the scheme and thus we can invoke the correctness of FE. We therefore have $y = C_{\mathsf{s},2^i}(\mathsf{SKE.ct_i}, \mathsf{K_i}, \mathsf{R_i}) = \widehat{\mathsf{FE}}.\mathsf{KeyGen}(\widehat{\mathsf{FE}}.\mathsf{msk_i}, \mathsf{D_{s,2^i}}[\mathsf{SKE.ct_i}]; \widehat{\mathsf{R}}_\mathsf{i})$ by the correctness of FE and $z = D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}](\mathsf{S_i}, \hat{\mathbf{x}}) = \widehat{\mathsf{M}}_{\mathsf{2^i}}(\hat{\mathbf{x}}) = \mathsf{M}(\mathbf{x})$ by the correctness of $\widehat{\mathsf{FE}}$ and SKE.

Security. Here, we prove that RGbNFA defined above is secure, if so is FE. Formally, we have the following theorem.

Theorem B.4.1. *Assume that* FE *satisfies full simulation based security,* $\widehat{\mathsf{FE}}$ *satisfies selective simulation security, and that* PRF *is a secure pseudorandom function. Then,* NfaABE *satisfies selective simulation security.*

Proof. Security follows analogously to that of Theorem B.2.1. The main difference is that the PE^+ simulator is replaced by the FE simulator and we additionally invoke the security of SKE to achieve the function privacy. Other differences are that we have to change $(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j)$ to be sampled using true randomness instead of K_j using the security of the PRF and we introduce the step where we undo the changes added during the hybrid games similarly to the proof of Theorem 3.6.4. Below, we give the hybrid games to prove the security. In more detail, we consider $\mathbf{Game}_i$ for $i \in [0, \lambda + 1]$ as follows.

$\mathbf{Game}_i$**:** The game proceeds as follows. In the following $\mathsf{FE}.\mathrm{Sim}$ is the simulator for FE.

Setup. At the beginning of the game, the challenger takes 1^λ as input and samples $\{\mathsf{K}_j, \widehat{\mathsf{K}}_j, \mathsf{R}_j, \mathsf{S}_j\}_{j\in[0,\lambda]}$. It also computes $(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{d}(\lambda)}, 1^{\mathsf{out}(\lambda)}; \mathsf{K}_j)$ for $j \in [0, \lambda]$.

The challenger answers the queries made by A as follows.

Garbling the NFA. A takes 1^λ as input and submits an NFA machine M of size s and a set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|X|}\} \subset \Sigma^*$ to the challenger. The challenger computes M_G as follows. The challenger computes $\mathsf{SKE.ct_j}$ and $\mathsf{FE.ct}_j$ for $j \in [0, \lambda]$ as

$$\mathsf{FE.ct_j} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk_j}, (\mathsf{SKE.ct_j}, \widehat{\mathsf{K}}_\mathsf{j}, \widehat{\mathsf{R}}_\mathsf{j})), \quad \mathsf{SKE.ct_j} \leftarrow \begin{cases} \mathsf{SKE.Enc}(\mathsf{S_j}, \mathsf{M}) & \text{If } \lambda \geq j \geq i \\ \mathsf{SKE.Enc}(\mathsf{S_j}, 0^\mathsf{s}) & \text{If } j \leq i-1, \end{cases}$$

and returns $M_G = (\{\mathsf{SKE.ct_j}, \mathsf{FE.mpk_j}, \mathsf{FE.ct_j}\}_{\mathsf{j}\in[0,\lambda]})$ to A.

Simulating Encodings. To generate an encoding for $\mathbf{x} \in X$, the challenger sets $j := \lceil \log |\mathbf{x}| \rceil$ and generates $(\widehat{\mathsf{FE}}.\mathsf{mpk}_j, \widehat{\mathsf{FE}}.\mathsf{msk}_j) = \widehat{\mathsf{FE}}.\mathsf{Setup}(1^\lambda, 1^{2^j\eta}, 1^{\hat{\mathsf{d}}(\lambda)}, 1^1; \widehat{\mathsf{K}}_j)$. It then computes

$$\widehat{\mathsf{FE}}.\mathsf{ct} \leftarrow \begin{cases} \widehat{\mathsf{FE}}.\mathsf{Enc}(\widehat{\mathsf{FE}}.\mathsf{mpk_j}, (\mathsf{S_j}, \hat{\mathbf{x}})) & \text{If } \lambda \geq j \geq i \\ \widehat{\mathsf{FE}}.\mathrm{Sim}(\widehat{\mathsf{FE}}.\mathsf{mpk}_j, \widehat{\mathsf{FE}}.\mathsf{sk}_{D_{\mathsf{s},2^j}[\mathsf{SKE.ct_j}]}, D_{\mathsf{s},2^j}[\mathsf{SKE.ct_j}], \mathsf{M}(\mathbf{x})) & \text{If } j \leq i-1 \end{cases}, \quad \text{(B.2)}$$

where $\widehat{\mathsf{FE}}.\mathsf{sk}_{D_{\mathsf{s},2^j}[\mathsf{SKE.ct_j}]} \leftarrow \widehat{\mathsf{FE}}.\mathsf{KeyGen}(\widehat{\mathsf{FE}}.\mathsf{msk}_j, D_{\mathsf{s},2^j}[\mathsf{SKE.ct_j}]; \widehat{\mathsf{R}}_\mathsf{j})$. It also computes $\mathsf{FE.sk}_j = \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_j, \mathsf{FE.msk}_j, C_{\mathsf{s},2^j}; \mathsf{R}_j)$. The encoding of $\mathbf{x}$ is $c = (\widehat{\mathsf{FE}}.\mathsf{mpk}_j, \mathsf{FE.sk}_j, \widehat{\mathsf{FE}}.\mathsf{ct})$.

Finally, A outputs its guess b'.

It is easy to see that $\mathbf{Game}_0$ is the same as $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGbNFA,A}}(1^\lambda)$. Furthermore, we can construct a simulator for RGbNFA from the challenger in $\mathbf{Game}_{\lambda+1}$ appropriately, since the challenger in $\mathbf{Game}_{\lambda+1}$ only uses s and $|\mathbf{x}|$ to simulate a garbled NFA and an encoding, respectively. Therefore, it suffices to show the indistinguishability between $\mathbf{Game}_i$ and $\mathbf{Game}_{i+1}$. To do so, we consider two cases separately depending on whether $i \leq i_{\max}$ or not, where $i_{\max}$ is defined as in the proof of Theorem 3.6.4.

We first consider the case of $i \geq i_{\max} + 1$. In this case, the only difference between $\mathbf{Game}_i$ and $\mathbf{Game}_{i+1}$ is the way $\mathsf{SKE.ct_i}$ is generated, since the upper branch of Equation (B.2) is never triggered when answering the encoding query because of the definition of $i_{\max}$. The indistinguishability of the two games immediately follows from the security of SKE since S_i is never used except when generating $\mathsf{SKE.ct_i}$.

We then consider the case of $i \leq i_{\max}$. The proof for this case closely follows the proof of Theorem B.2.1 except for we some changes that we explained at the beginning of the proof.

$\mathbf{Game}_{i,0}$: The game is the same as $\mathbf{Game}_i$.

$\mathbf{Game}_{i,1}$: The game is the same as the previous game except that $\mathsf{FE.sk}_i = \mathsf{FE.KeyGen}(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i, C_{\mathsf{s},2^i}; \mathsf{R}_i)$ and $(\widehat{\mathsf{FE}}.\mathsf{mpk}_i, \widehat{\mathsf{FE}}.\mathsf{msk}_i) = \widehat{\mathsf{FE}}.\mathsf{Setup}(1^\lambda, 1^{2^i \eta}, 1^{\hat{\mathsf{d}}(\lambda)}, 1^1; \widehat{\mathsf{K}}_i)$ are computed at the setup phase.

$\mathbf{Game}_{i,2}$: The game is the same as the previous game except that $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ and $\mathsf{FE.sk}_i$ are generated using true randomness instead of using the PRF keys.

$\mathbf{Game}_{i,3}$: In this game, to answer a garbling query, $\mathsf{FE.ct_i}$ is computed as

$$\mathsf{FE.ct_i} \leftarrow \mathsf{FE}.\mathrm{Sim}(\mathsf{FE.mpk_i}, \mathsf{FE.sk_i}, \mathsf{C_{s,2^i}}, \widehat{\mathsf{FE}}.\mathsf{sk}_{\mathsf{D_{s,2^i}[SKE.ct_i]}}, 1^{\mathsf{inp}(\lambda)})$$

where $\widehat{\mathsf{FE}}.\mathsf{sk}_{D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}]} \leftarrow \widehat{\mathsf{FE}}.\mathsf{KeyGen}(\widehat{\mathsf{FE}}.\mathsf{msk}_i, D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}]; \widehat{\mathsf{R}}_\mathsf{i})$.

$\mathbf{Game}_{i,4}$: In this game, to answer a garbling query, $(\widehat{\mathsf{FE}}.\mathsf{mpk}_i, \widehat{\mathsf{FE}}.\mathsf{msk}_i)$ and $\widehat{\mathsf{FE}}.\mathsf{sk}_{D_{\mathsf{s},2^i}[\mathsf{SKE.ct_i}]}$ are generated using true randomness instead of using the PRF keys $\widehat{\mathsf{K}}_i$ and $\widehat{\mathsf{R}}_i$.

$\mathbf{Game}_{i,5}$: In this game, to answer an encoding query, we generate

$$\widehat{\mathsf{FE}}.\mathsf{ct} \leftarrow \widehat{\mathsf{FE}}.\mathrm{Sim}(\widehat{\mathsf{FE}}.\mathsf{mpk_i}, \widehat{\mathsf{FE}}.\mathsf{sk}_{\mathsf{D_{s,2^i}[SKE.ct_i]}}, \mathsf{D_{s,2^i}[SKE.ct_i]}, \mathsf{M}(\mathbf{x}))$$

instead of honestly generating it.

$\mathbf{Game}_{i,6}$: In this game, $\mathsf{SKE.ct_i}$ is changed to be $\mathsf{SKE.Enc}(\mathsf{S_i}, 0^{|\mathsf{M}|})$.

$\mathbf{Game}_{i,7}$: In this game, we undo the changes we added from $\mathbf{Game}_{i,0}$ to $\mathbf{Game}_{i,4}$. Namely, we generate $(\widehat{\mathsf{FE}}.\mathsf{mpk}_i, \widehat{\mathsf{FE}}.\mathsf{msk}_i)$ by using $\widehat{\mathsf{K}}_i, \widehat{\mathsf{R}}_i$, $\mathsf{FE.ct_i}$ by honestly encrypting $(\mathsf{SKE.ct_i}, \widehat{\mathsf{K}}_\mathsf{i}, \widehat{\mathsf{R}}_\mathsf{i})$ (but $\mathsf{SKE.ct_i}$ is still an encryption of $0^{|M|}$), and $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ by using $\mathsf{K}_i, \mathsf{R}_i$.

The indistinguishability between $\mathbf{Game}_{i,j-1}$ and $\mathbf{Game}_{i,j}$ for $j \in [5]$ follows similarly to the proof of Theorem B.2.1, except that we use the security of $\widehat{\mathsf{FE}}$ rather than PE^+ when moving from $\mathbf{Game}_{i,4}$ to $\mathbf{Game}_{i,5}$. The indistinguishability between $\mathbf{Game}_{i,5}$ and $\mathbf{Game}_{i,6}$ follows from the security of SKE, since S_i is never used except when generating $\mathsf{SKE.ct_i}$ in these games even if the upper branch of Equation (B.2) is triggered, due to the change we added in $\mathbf{Game}_{i,5}$. The indistinguishability between $\mathbf{Game}_{i,6}$ and $\mathbf{Game}_{i,7}$ can be shown by repeating the same argument for showing $\mathbf{Game}_{i,0} \overset{c}{\approx} \mathbf{Game}_{i,4}$ in the reverse order. Finally, we note that $\mathbf{Game}_{i,7}$ is equivalent to $\mathbf{Game}_{i+1}$. These imply that $\mathbf{Game}_i$ and $\mathbf{Game}_{i+1}$ are indistinguishable, which completes the proof of the theorem. $\square$

Efficiency. In the above construction, the efficiency requirement (Definition B.3.3) is not satisfied, since the encoding algorithm constructs $C_{\mathsf{s},2^i}$ whose size is polynomially dependent on the size of NFA M. This is problematic when $|M| \gg |\mathbf{x}|$. To resolve the issue, we use the same idea as that we used in Sec. 3.7. Namely, we combine our RGC construction above with the one that poses upper-bound on the input length [Goldwasser *et al.* (2013*a*)]. Namely, when we garble an NFA with size $|M|$, we convert M into an equivalent circuit $\widehat{M}_{|M|,j}$ with input length j for all $j \in [|M|]$ and then garble all of them using [Goldwasser *et al.* (2013*a*)]. In addition, we garble M with the above scheme with $\mathsf{s} = |M|$, which supports unbounded length. We do this by deriving randomness from a single PRF key and therefore gsk is compact. To encode $\mathbf{x}$, we encode it with the above scheme with different s in parallel. Namely, we encode $\mathbf{x}$ by the above scheme with all of $\mathsf{s} \leq [|\mathbf{x}|]$. Furthermore, we also choose $|\mathbf{x}|$-th instance of the RGC of [Goldwasser *et al.* (2013*a*)] and encodes it. Similarly to the construction in Sec. 3.7, evaluation algorithm first sees if $|\mathbf{x}| > |M|$ and uses the above scheme to decode if so. Otherwise, it uses the bounded input scheme [Goldwasser *et al.* (2013*a*)]. It can be seen that the encoding algorithm runs in polynomial time in $|\mathbf{x}|$ independent of $|M|$. Furthermore, the security of the scheme is preserved, since the construction simply runs secure schemes in parallel.

Generalizing to Bounded Keys. We note that by replacing the inner scheme with a bounded key FE scheme [Gorbunov *et al.* (2012); Agrawal and Rosen (2017)], the construction immediately generalizes to support bounded number of (reusable) garbled circuits.

APPENDIX C

Appendices for Chapter 4

C.1 Instantiating the Ingredients

Here, we instantiate the necessary ingredients for our construction, namely ABE schemes for the relations R^{MUKP} (i.e., multi-use key-policy unbounded ABE with polynomial valued attributes) and R^{MUCP} (i.e., multi-use ciphertext-policy unbounded ABE with polynomial valued attributes). For both key-policy and ciphertext-policy cases, we essentially use schemes from [Chen *et al.* (2018)], but with the modification that we allow unbounded multi-use of the same attribute in an MSP, which is essential for our purpose. Due to this modification, we can no longer prove *the adaptive* security of the scheme from the MDDH_k assumption as was done by [Chen *et al.* (2018)]. However, we can still prove *semi-adaptive* security from the same assumption for the key-policy case and *selective** security from the DLIN assumption for the ciphertext-policy case (please see Section 4.7.3 for the definitions).

C.1.1 Preliminaries

Here, we recap necessary notations and definitions for this section following [Chen *et al.* (2018)].

Notation on Bilinear Maps. A bilinear group generator takes as input 1^λ and outputs a group description $\mathbb{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where p is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are cyclic groups of order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map. We require that the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ as well as the bilinear map e can be efficiently computed. We employ the implicit representation of group elements: for a matrix $\mathbf{A}$ over $\mathbb{Z}_p$, we define $[\mathbf{A}]_1 := g_1^{\mathbf{A}}$, $[\mathbf{A}]_2 := g_2^{\mathbf{A}}$, $[\mathbf{A}]_T := g_T^{\mathbf{A}}$, where exponentiation is carried out component-wise. We also let $e([\mathbf{A}]_1, [\mathbf{B}]_2) = [\mathbf{A}\mathbf{B}]_T$ for $[\mathbf{A}]_1$ and $[\mathbf{B}]_2$.

Here, we define the decisional linear assumption (DLIN) and the MDDH_k assumption.

Definition C.1.1 (Decisional linear assumption.)**.** We say that the DLIN assumption holds on $\mathbb{G}$ if we have

$$(\mathbb{G}, [x_1]_1, [x_2]_1, [x_1 y_1]_1, [x_2 y_2]_1, [y_1 + y_2]_2) \approx_c (\mathbb{G}, [x_1]_1, [x_2]_1, [x_1 y_1]_1, [x_2 y_2]_1, [\Phi]_2)$$

for $x_1, x_2, y_1, y_2 \leftarrow \mathbb{Z}_p$ and $\Phi \leftarrow \mathbb{Z}_p$.

Definition C.1.2. Let $k \geq 1$ be an integer. We say that the MDDH_k assumption holds on $\mathbb{G}_1$ if we have

$$(\mathbb{G}, [\mathbf{B}]_1, [\mathbf{Bs}]_1) \approx_c (\mathbb{G}, [\mathbf{B}]_1, [\mathbf{t}]_1)$$

for $\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$, $\mathbf{s} \leftarrow \mathbb{Z}_p^k$, and $\mathbf{t} \leftarrow \mathbb{Z}_p^{k+1}$.

The MDDH_k assumption on $\mathbb{G}_2$ can be defined in an analogous way. As [Escala *et al.* (2017)] showed, the MDDH_k assumption on a group is implied by the k-Lin assumption on the same group.

We also recall the following statistical lemma.

Lemma C.1.3 (Adapted from Lemma 1 in [Chen *et al.* (2018)])**.** *Let* $\mathbf{L} := \mathbb{Z}_p^{\ell \times m}$ *be a matrix and* $\{\delta_j \in \{0,1\}\}_{j\in[\ell]}$ *be a set of binary integers such that the vector* $(1, 0, \ldots, 0)^\top$ *is not in* $\mathrm{span}(\{\mathbf{L}_j^\top\}_{j:\delta_j=1})$*. Then, the following distributions are the same:*

$$\{(0\|\mathbf{k}')\mathbf{L}_j^\top + r_j\delta_j\}_{j\in[\ell]} \text{ and } \{(1\|\mathbf{k}')\mathbf{L}_j^\top + r_j\delta_j\}_{j\in[\ell]},$$

where $\mathbf{k}' \leftarrow \mathbb{Z}_p^{m-1}$ *is a row vector and* $r_j \leftarrow \mathbb{Z}_p$*.*

C.1.2 The Construction of Ingredient KP-ABE

Here, we provide an ABE scheme for R^{MUKP}, denoted by kpABE. The construction is essentially the same as the unbounded KP-ABE given in [Chen *et al.* (2018)] with the modification that we allow unbounded multi-use of the same attribute in an MSP.

$\mathsf{Setup}(1^\lambda)$**:** On input 1^λ, sample

$$\mathbf{A}_1 \leftarrow \mathbb{Z}_p^{(2k+1)\times k}, \mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}, \mathbf{W}, \mathbf{W}_0, \mathbf{W}_1 \leftarrow \mathbb{Z}_p^{(2k+1)\times(k+1)}, \mathbf{k} \leftarrow \mathbb{Z}_p^{2k+1}$$

and output

$$\mathsf{mpk} := \big([\mathbf{A}_1^\top, \mathbf{A}_1^\top \mathbf{W}, \mathbf{A}_1^\top \mathbf{W}_0, \mathbf{A}_1^\top \mathbf{W}_1]_1, e([\mathbf{A}_1^\top]_1, [\mathbf{k}]_2)\big) \in \mathbb{G}_1^{k\times(2k+1)} \times (\mathbb{G}_1^{k\times(k+1)})^3 \times \mathbb{G}_T^k$$

and

$$\mathsf{msk} := (\mathbf{k}, \mathbf{B}, \mathbf{W}, \mathbf{W}_0, \mathbf{W}_1).$$

$\mathsf{Enc}(\mathsf{mpk}, (\mathsf{S}, 1^{\mathsf{s}_{\max}}), \mu)$**:** On input an attribute set $S = \{s_1, \ldots, s_\ell\} \subset \mathbb{Z}$, and $\mu \in \mathbb{G}_T$, pick $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1)$ for $s \in S$ and output

$$\mathsf{ct}_{\mathsf{S}} := \left(\begin{array}{c} C_0 = [\mathbf{c}^\top]_1,\ C := e([\mathbf{c}]^\top, [\mathbf{k}]_2) \cdot \mu, \\ \big\{ C_{1,s} := [\mathbf{c}^\top \mathbf{W} + \mathbf{c}_s^\top (\mathbf{W}_0 + s\mathbf{W}_1)]_1,\ C_{2,s} := [\mathbf{c}_s^\top]_1 \big\}_{s\in S} \end{array} \right).$$

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{mpk}, ((\mathbf{L}, \rho), 1^{\rho_{\max}}))$**:** On input a monotone span program $(\mathbf{L} \in \mathbb{Z}_p^{\ell\times m}, \rho)$, pick $\mathbf{K}' \leftarrow \mathbb{Z}_p^{(2k+1)\times(m-1)}$, $\mathbf{d}_j \leftarrow \mathrm{span}(\mathbf{B})$ for all $j \in [\ell]$ and output

$$\mathsf{sk}_{(\mathbf{L},\rho)} := \left(\left\{ \begin{array}{c} K_{0,j} := [(\mathbf{k} \| \mathbf{K}')\mathbf{L}_j^\top + \mathbf{W}\mathbf{d}_j]_2, \\ K_{1,j} := [\mathbf{d}_j]_2,\ K_{2,j} := [(\mathbf{W}_0 + \rho(j)\mathbf{W}_1)\mathbf{d}_j]_2 \end{array} \right\}_{j\in[\ell]} \right),$$

where $\mathbf{L}_j$ is the j-th row of $\mathbf{L}$.

$\mathsf{Dec}(\mathsf{mpk}, \mathsf{ct}, (\mathsf{S}, 1^{\mathsf{s}_{\max}}), \mathsf{sk}_{(\mathbf{L},\rho)}, ((\mathbf{L}, \rho), 1^{\rho_{\max}}))$**:** Since S satisfies $(\mathbf{L}, \rho)$, one can compute $\{\omega_j\}$ such that

$$\sum_{j:\rho(j)\in S} \omega_j \mathbf{L}_j = (1, 0, \ldots, 0).$$

Then, compute

$$K = \prod_{j:\rho(j)\in S} \big(e(C_0, K_{0,j}) e(C_{1,\rho(j)}, K_{1,j})^{-1} e(C_{2,\rho(j)}, K_{2,j}) \big)^{\omega_j}$$

and retrieve the message by C/K.

Correctness. For j such that $\rho(j) \in S$, we have

$$\begin{aligned}
&e(C_0, K_{0,j})e(C_{1,\rho(j)}, K_{1,j})^{-1}e(C_{2,\rho(j)}, K_{2,j}) \\
= \;& e([\mathbf{c}^\top]_1, [(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top + \mathbf{W}\mathbf{d}_j]_2) \cdot e([\mathbf{c}^\top\mathbf{W} + \mathbf{c}_j^\top(\mathbf{W}_0 + \rho(j)\mathbf{W})]_1, [\mathbf{d}_j]_2)^{-1} \\
&\cdot e([\mathbf{c}_j^\top], [(\mathbf{W}_0 + \rho(j)\mathbf{W}_1)\mathbf{d}_j]_2) \\
= \;& [\mathbf{c}^\top(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top]_T.
\end{aligned}$$

The correctness readily follows from the following equation.

$$K = \prod_{j:\rho(j)\in S} [\omega_j \mathbf{c}^\top(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top]_T = [\mathbf{c}^\top(\mathbf{k}\|\mathbf{K}') \sum_{j:\rho(j)\in S} \omega_j \mathbf{L}_j^\top]_T = [\mathbf{c}^\top\mathbf{k}]_T.$$

C.1.3 Security Proof

Here, we prove the semi-adaptive security of the construction in Appendix C.1.2. To do so, we first recall a special case of the prime-order entropy expansion lemma from [Chen *et al.* (2018)].

Lemma C.1.4 (Lemma 12 from [Chen *et al.* (2018)]). *Pick basis* $(\mathbf{A}_1, \mathbf{a}_2, \mathbf{A}_3) \leftarrow \mathbb{Z}_p^{(2k+1)\times k} \times \mathbb{Z}_p^{2k+1} \times \mathbb{Z}_p^{(2k+1)\times k}$ *and define its dual* $(\mathbf{A}_1^\|, \mathbf{a}_2^\|, \mathbf{A}_3^\|)$ *such that* $\mathbf{A}_i^\top \mathbf{A}_j = \mathbf{I}$ *if* $i = j$ *and* $\mathbf{A}_i^\top \mathbf{A}_j = \mathbf{0}$ *otherwise, where we set* $\mathbf{A}_2 := \mathbf{a}_2$*. With* $\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$ *and for any polynomially bounded* $n \in \mathbb{N}$*, we have*

$$\left\{\begin{array}{rc}
\mathsf{aux}: & [\mathbf{A}_1^\top]_1, [\mathbf{A}_1^\top\mathbf{W}]_1, [\mathbf{A}_1^\top\mathbf{W}_0]_1, [\mathbf{A}_1^\top\mathbf{W}_1]_1 \\
\mathsf{ct}: & [\mathbf{c}^\top]_1, [\mathbf{c}^\top\mathbf{W} + \mathbf{c}_s^\top(\mathbf{W}_0 + s\mathbf{W}_1)]_1, [\mathbf{c}_s^\top]_1 \\
\mathsf{sk}: & [\mathbf{W}\mathbf{D}_s]_2, [\mathbf{D}_s]_2, [(\mathbf{W}_0 + s\mathbf{W}_1)\mathbf{D}_s]_2
\end{array}\right\}_{s\in[n]}$$

$$\overset{c}{\approx} \left\{\begin{array}{rc}
\mathsf{aux}: & [\mathbf{A}_1^\top]_1, [\mathbf{A}_1^\top\mathbf{W}]_1, [\mathbf{A}_1^\top\mathbf{W}_0]_1, [\mathbf{A}_1^\top\mathbf{W}_1]_1 \\
\mathsf{ct}: & [\boxed{\mathbf{c}}^\top]_1, [\boxed{\mathbf{c}}^\top(\mathbf{W} + \boxed{\mathbf{V}_s^{(2)}}) + \boxed{\mathbf{c}_s}^\top(\mathbf{W}_0 + s\mathbf{W}_1 + \boxed{\mathbf{U}_s^{(2)}})]_1, [\boxed{\mathbf{c}_s}^\top]_1 \\
\mathsf{sk}: & [(\mathbf{W} + \boxed{\mathbf{V}_s^{(2)}})\mathbf{D}_s]_2, [\mathbf{D}_s]_2, [(\mathbf{W}_0 + s\mathbf{W}_1 + \boxed{\mathbf{U}_s^{(2)}})\mathbf{D}_s]_2
\end{array}\right\}_{s\in[n]},$$

under the MDDH_k *assumption on* $\mathbb{G}_1$ *and* $\mathbb{G}_2$*, where* $\mathbf{W}, \mathbf{W}_0, \mathbf{W}_1 \leftarrow \mathbb{Z}_p^{(2k+1)\times(k+1)}$*,* $\mathbf{U}_s^{(2)}, \mathbf{V}_s^{(2)} \leftarrow \mathrm{span}^{k+1}(\mathbf{a}_2^\|)$*,* $\mathbf{D}_s \leftarrow \mathrm{span}^{k+1}(\mathbf{B})$*, and* $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1)$ *in the left distribution while* $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{a}_2)$ *in the right distribution.*

We then state the following theorem. The proof is similar to that of [Chen *et al.* (2018)], but since certain information theoretic step in [Chen *et al.* (2018)] does not work in the multi-use setting, we modify the proof so that we decompose the secret key into smaller pieces and gradually change their distributions by a carefully chosen sequence of hybrid games. Since it is essential for the simulator to know the challenge attribute S in these hybrid games, we can only prove semi-adaptive security instead of adaptive security.

Theorem C.1.5. *The ABE scheme for relation R^{MUKP} (i.e., multi-use key-policy unbounded ABE with polynomial valued attributes) in Appendix C.1.2 is semi-adaptively secure under the* MDDH_k *assumption.*

Proof. To prove the theorem, we define various forms of ciphertext (of message μ under attribute S).

Normal: A normal ciphertext is generated by Enc. In particular, $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1)$.

E-normal: This is the same as normal ciphertext except that $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{a}_2)$ and we use the following substitution:

$$\mathbf{W} \mapsto \hat{\mathbf{V}}_s := \mathbf{W}+\mathbf{V}_s^{(2)} \text{ in the } s\text{-th component and } \mathbf{W}_0+s\mathbf{W}_1 \mapsto \hat{\mathbf{U}}_s := \mathbf{W}_0+s\mathbf{W}_1+\mathbf{U}_s^{(2)}$$

where $\mathbf{U}_s^{(2)}, \mathbf{V}_s^{(2)} \leftarrow \mathrm{span}^{k+1}(\mathbf{a}_2^{\|})$. Concretely, an E-normal ciphertext is of the form

$$\mathsf{ct}_\mathsf{S} := \left([\mathbf{c}^\top]_1,\ \left\{[\mathbf{c}^\top \boxed{\hat{\mathbf{V}}_s} + \mathbf{c}_\mathsf{s}^\top \boxed{\hat{\mathbf{U}}_s}]_1,\ [\mathbf{c}_\mathsf{s}^\top]_1\right\}_{\mathsf{s}\in\mathsf{S}},\ \mathsf{e}([\mathbf{c}]^\top, [\mathbf{k}]_2) \cdot \mu \right),$$

where $\boxed{\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{a}_2)}$.

We then define various forms of keys (for span program $\mathbf{L}$).

Normal. A normal key is generated by KeyGen.

E-normal: An E-normal key $\mathsf{sk}_{\mathbf{L},\rho} = \{K_{0,j}, K_{1,j}, K_{2,j}\}_{j\in[\ell]}$ is sampled as

$$\mathsf{sk}_{\mathbf{L},\rho} := \left(\left\{[(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top + \boxed{\hat{\mathbf{V}}_{\rho(j)}}\mathbf{d}_j]_2,\ [\mathbf{d}_j]_2,\ [\boxed{\hat{\mathbf{U}}_{\rho(j)}}\mathbf{d}_j]_2\right\}_{j\in[\ell]}\right).$$

Here, $\mathbf{d}_i \leftarrow \mathrm{span}(\mathbf{B})$ and $\mathbf{K}' \leftarrow \mathbb{Z}_p^{(2k+1)\times(m-1)}$ are sampled freshly for every key generation. On the other hand, we use the same $\hat{\mathbf{U}}_s$ and $\hat{\mathbf{V}}_s$ that are used when generating the E-normal challenge ciphertext.

SF: An SF key $\mathsf{sk}_{\mathbf{L},\rho} = \{K_{0,j}, K_{1,j}, K_{2,j}\}_{j\in[\ell]}$ is sampled as

$$(K_{0,j}, K_{1,j}, K_{2,j}) :=$$
$$\begin{cases} \left([(\mathbf{k} + \boxed{\alpha\mathbf{a}_2^{\|}} \| \mathbf{K}')\mathbf{L}_j^{\top} + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2\right) & \text{If } \rho(j) \in S \\ \left([(\mathbf{k} + \boxed{\alpha\mathbf{a}_2^{\|}} \| \mathbf{K}')\mathbf{L}_j^{\top} + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j + \boxed{r_j\mathbf{a}_2^{\|}}]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2\right) & \text{If } \rho(j) \notin S \end{cases}$$

where $\boxed{r_j \leftarrow \mathbb{Z}_p}$, $\mathbf{d}_j \leftarrow \mathrm{span}(\mathbf{B})$, $\mathbf{K}' \leftarrow \mathbb{Z}_p^{(2k+1)\times(m-1)}$ and S is the attribute associated with the challenge ciphertext. We note that S is well-defined when generating a secret key because we are in the semi-adaptive security game. We sample fresh $\mathbf{d}_j$ and r_j for every key generation, while we use the same $\alpha \leftarrow \mathbb{Z}_p$ throughout the game. We also note that we use the same $\hat{\mathbf{U}}_s$ and $\hat{\mathbf{V}}_s$ that are used for generating the E-normal challenge ciphertext.

We define the following sequence of games to prove the security. Let the number of key generation queries made by an adversary be q.

$\mathbf{Game}_0$**:** This is the real security game for semi-adaptive security where all ciphertexts and keys are normal.

$\mathbf{Game}_{0'}$: In this game, we change the challenge ciphertext and all keys to be E-normal ones.

$\mathbf{Game}_{i^\star}$**:** In this game, the challenge ciphertext and the first $i^\star - 1$ secret keys given to the adversary are SF, while rest of the secret keys are E-normal.

$\mathbf{Game}_{\mathrm{Final}}$**:** This is the same as $\mathbf{Game}_{q+1}$ except that the challenge ciphertext is a E-normal one for a random message in $\mathbb{G}_T$.

Let us fix a PPT adversary $\mathcal{A}$ and denote the advantage of $\mathcal{A}$ in $\mathbf{Game}_{\mathrm{xx}}$ by A_{xx}. We can easily see that $\mathbf{Game}_{0'} = \mathbf{Game}_1$ and $\mathsf{A}_{\mathrm{Final}} = 0$. Therefore, to complete the proof of Theorem C.1.5, it suffices to prove Lemma C.1.6, C.1.7, and C.1.8 in the following. $\square$

Lemma C.1.6. *Under the* MDDH_k *assumption on* $\mathbb{G}_1$ *and* $\mathbb{G}_2$*, we have* $|\mathsf{A}_0 - \mathsf{A}_{0'}| = \mathrm{negl}(\lambda)$.

Proof. For the sake of contradiction, we assume that $\mathcal{A}$ distinguishes $\mathbf{Game}_0$ and $\mathbf{Game}_{0'}$ with non-negligible advantage and show that we can construct another adversary $\mathcal{B}$ that distinguishes the two distributions in Lemma C.1.4 with the same advantage. By the same lemma, this implies an adversary against MDDH_k with non-negligible advantage. Let n be the upper bound on the running time of $\mathcal{A}$. On input

$$\left\{\begin{array}{ll} \mathsf{aux}: & [\mathbf{A}_1^\top]_1, [\mathbf{A}_1^\top\mathbf{W}]_1, [\mathbf{A}_1^\top\mathbf{W}_0]_1, [\mathbf{A}_1^\top\mathbf{W}_1]_1 \\ \mathsf{ct}: & [\mathbf{C}_0]_1, [\mathbf{C}_{1,s}]_1, [\mathbf{C}_{2,s}]_1 \\ \mathsf{sk}: & [\mathbf{K}_{0,s}]_2, [\mathbf{K}_{1,s}]_2, [\mathbf{K}_{2,s}]_2 \end{array}\right\}_{s\in[n]},$$

$\mathcal{B}$ proceeds as follows.

Setup. It samples $\mathbf{k} \leftarrow \mathbb{Z}_p^{2k+1}$ and give $\mathsf{mpk} := (\mathsf{aux}, e([\mathbf{A}_1^\top]_1, [\mathbf{k}]_2))$ to $\mathcal{A}$. Then, $\mathcal{A}$ declares its target $(S, 1^{s_{\max}})$ to $\mathcal{B}$.

Ciphertext. When $\mathcal{A}$ asks for the challenge ciphertext with respect to messages (μ_0, μ_1), $\mathcal{B}$ samples $\beta \leftarrow \{0,1\}$ and sets the challenge ciphertext as

$$\mathsf{ct}_\mathsf{S} := \{[\mathbf{C}_0]_1, \{[\mathbf{C}_{1,\mathsf{s}}]_1, [\mathbf{C}_{2,\mathsf{s}}]_1\}_{\mathsf{s}\in\mathsf{S}}, \mathsf{e}([\mathbf{C}_0]_1, [\mathbf{k}]_2)\cdot\mu_\beta\}\,.$$

Note that since $n \geq s_{\max} = \max_{s\in S}|s|$, $\mathcal{A}$ can simulate the challenge ciphertext using the given terms.

Secret Keys. When $\mathcal{A}$ asks for the secret key for $((\mathbf{L} \in \mathbb{Z}_p^{\ell\times m}, \rho), 1^{\rho_{\max}})$, $\mathcal{B}$ samples $\mathbf{K}' \leftarrow \mathbb{Z}_p^{(2k+1)\times(m-1)}$ and $\tilde{\mathbf{d}}_j \leftarrow \mathbb{Z}_p^{k+1}$ for $j \in [\ell]$ and sets

$$\mathsf{sk}_{(\mathbf{L},\rho)} := \left\{[(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top + \mathbf{K}_{0,\rho(j)}\tilde{\mathbf{d}}_j]_2,\ [\mathbf{K}_{1,\rho(j)}\tilde{\mathbf{d}}_j]_2,\ [\mathbf{K}_{2,\rho(j)}\tilde{\mathbf{d}}_j]_2\right\}_{j\in[\ell]},$$

where we implicitly set $\mathbf{d}_j := \mathbf{D}_{\rho(j)}\tilde{\mathbf{d}}_j$, which is uniformly distributed over $\mathrm{span}(\mathbf{B})$. Note that since $n \geq \rho_{\max} = \max_{j\in[\ell]}|\rho(j)|$, $\mathcal{A}$ can simulate the challenge ciphertext using the given terms.

Guess. When $\mathcal{A}$ halts with output β', $\mathcal{B}$ outputs 1 if $\beta' = \beta$ and 0 otherwise.

Observe that when $\mathcal{B}$'s input is from the left distribution in Lemma C.1.4, it simulates $\mathbf{Game}_0$ and when it is the right distribution, it simulates $\mathbf{Game}_{0'}$. This completes the proof of Lemma C.1.6. $\square$

Lemma C.1.7. *We have* $|\mathsf{A}_{q+1} - \mathsf{A}_{\mathrm{Final}}| = \mathrm{negl}(\lambda)$ *unconditionally.*

Proof. Let us fix all the randomness used in the games except for $\mathbf{k} \leftarrow \mathbb{Z}_p^{2k+1}$ and $\alpha \leftarrow \mathbb{Z}_p$. We set $\tilde{\mathbf{k}} := \mathbf{k} + \alpha \mathbf{a}_2^{\|}$ and show that the view of the adversary except for the challenge ciphertext can be simulated by $\tilde{\mathbf{k}}$. Namely, we show that the information of α (or equivalently, $\mathbf{k}$) is not used during the simulation, except for the challenge phase.

Setup. The only place where $\mathbf{k}$ is used in the generation of master public key is in the computation of the term $e([\mathbf{A}_1^\top]_1, [\mathbf{k}]_2)$. However, this term can be simulated by $\tilde{\mathbf{k}}$ instead, since we have

$$e([\mathbf{A}_1^\top]_1, [\tilde{\mathbf{k}}]_2) = e([\mathbf{A}_1^\top]_1, [\mathbf{k} + \alpha \mathbf{a}_2^{\|}]_2) = e([\mathbf{A}_1^\top]_1, [\mathbf{k}]_2).$$

Secret Keys. Then, we observe that any secret key $\mathsf{sk}_{\mathbf{L},\rho} = \{K_{0,j}, K_{1,j}, K_{2,j}\}_{j \in [\ell]}$ generated during the game can be represented as

$$(K_{0,j}, K_{1,j}, K_{2,j}) := \begin{cases} \left([(\tilde{\mathbf{k}} \| \mathbf{K}')\mathbf{L}_j^\top + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j + r_j \mathbf{a}_2^{\|}]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2 \right) & \text{If } \rho(j) \notin S \\ \left([(\tilde{\mathbf{k}} \| \mathbf{K}')\mathbf{L}_j^\top + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2 \right) & \text{If } \rho(j) \in S \end{cases}.$$

Namely, they can be simulated only from $\tilde{\mathbf{k}}$.

Next, we investigate the distribution of the challenge ciphertext.

Ciphertext. Recall that the challenge ciphertext consists of the components $[\mathbf{c}^\top]_1$, $[\mathbf{c}^\top \hat{\mathbf{V}}_s + \mathbf{c}_s^\top \hat{\mathbf{U}}_s]_1$, and $e([\mathbf{c}]^\top, [\mathbf{k}]_2) \cdot \mu_\beta$, where β is the challenge bit chosen by the challenger. Let us assume that $\mathbf{c} \notin \mathrm{span}(\mathbf{A}_1)$, since it occurs with probability $1 - 1/p$. Then we show that the last component of the challenge ciphertext is uniformly at random over $\mathbb{G}_T$. To see this, we observe

$$e([\mathbf{c}]^\top, [\mathbf{k}]_2) = e([\mathbf{c}^\top], [\tilde{\mathbf{k}}]_2) \cdot \boxed{e([\mathbf{c}^\top], [\mathbf{a}_2^{\|}])^\alpha},$$

where the boxed term above is distributed uniformly at random over $\mathbb{G}_T$ since $\mathbf{c}^\top \mathbf{a}_2^{\|} \neq \mathbf{0}$ and the information of α is not used anywhere else in the game. Therefore, the view of

$\mathbf{Game}_{q+1}$ is exactly the same as that of $\mathbf{Game}_{\mathrm{Final}}$, where random message on $\mathbb{G}_T$ is encrypted. This completes the proof of Lemma C.1.7. $\square$

Lemma C.1.8. *Under the* MDDH_k *assumption on* $\mathbb{G}_2$*, we have* $|\mathsf{A}_{i^\star} - \mathsf{A}_{i^\star+1}| = \mathrm{negl}(\lambda)$ *for* $i^\star \in [q]$.

Proof. In order to prove Lemma C.1.8, we further consider the following hybrid games. Let the $i^\star$-th key extraction query made by $\mathcal{A}$ be $((\mathbf{L} \in \mathbb{Z}_p^{\ell \times m}, \rho), 1^{\rho_{\max}})$.

$\mathbf{Game}_{i^\star,j^\star,1}$**:** This is the same as $\mathbf{Game}_{i^\star}$, except that the secret key $\mathsf{sk}_{\mathbf{L},\rho} = \{K_{0,j}, K_{1,j}, K_{2,j}\}_{j\in[\ell]}$ for the $i^\star$-th key extraction query is sampled as

$$(K_{0,j}, K_{1,j}, K_{2,j}) := \begin{cases} \left([(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j + \boxed{r_j\mathbf{a}_2^{\|}}]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2\right) & \text{If } j \le j^\star - 1 \wedge \rho(j) \notin S \\ \left([(\mathbf{k}\|\mathbf{K}')\mathbf{L}_j^\top + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2\right) & \text{If } j \ge j^\star \vee \rho(j) \in S \end{cases}$$

where $\mathbf{d}_j \leftarrow \mathrm{span}(\mathbf{B})$ is freshly sampled. It can be seen that the distribution of the key in this game is a hybrid between that of an SF key and an E-normal key.

$\mathbf{Game}_{i^\star,j^\star,2}$**:** This game is the same as $\mathbf{Game}_{i^\star,j^\star,1}$ except that to sample the $j^\star$-th component $(K_{0,j^\star}, K_{1,j^\star}, K_{2,j^\star})$ of the $i^\star$-th secret key, we sample $\boxed{\mathbf{d}_{j^\star} \leftarrow \mathbb{Z}_p^{k+1}}$ instead of $\mathbf{d}_{j^\star} \leftarrow \mathrm{span}(\mathbf{B})$.

$\mathbf{Game}_{i^\star,j^\star,3}$**:** This game is the same as $\mathbf{Game}_{i^\star,j^\star,2}$, except that $j^\star$-th component $(K_{0,j^\star}, K_{1,j^\star}, K_{2,j^\star})$ of the $i^\star$-th secret key is sampled as

$$(K_{0,j^\star}, K_{1,j^\star}, K_{2,j^\star}) := \begin{cases} \left([(\mathbf{k}\|\mathbf{K}')\mathbf{L}_{j^\star}^\top + \hat{\mathbf{V}}_{\rho(j^\star)}\mathbf{d}_{j^\star} + \boxed{r_{j^\star}\mathbf{a}_2^{\|}}]_2,\ [\mathbf{d}_{j^\star}]_2\ [\hat{\mathbf{U}}_{\rho(j^\star)}\mathbf{d}_{j^\star}]_2\right) & \text{If } \rho(j^\star) \notin S \\ \left([(\mathbf{k}\|\mathbf{K}')\mathbf{L}_{j^\star}^\top + \hat{\mathbf{V}}_{\rho(j^\star)}\mathbf{d}_{j^\star}]_2,\ [\mathbf{d}_{j^\star}]_2,\ [\hat{\mathbf{U}}_{\rho(j^\star)}\mathbf{d}_{j^\star}]_2\right) & \text{If } \rho(j^\star) \in S \end{cases},$$

where $r_{j^\star} \leftarrow \mathbb{Z}_p$, $\mathbf{d}_{j^\star} \leftarrow \mathbb{Z}_p^{k+1}$.

$\mathbf{Game}_{i^\star,j^\star,4}$**:** This game is the same as $\mathbf{Game}_{i^\star,j^\star,3}$, except that to sample the $j^\star$-th component $(K_{0,j^\star}, K_{1,j^\star}, K_{2,j^\star})$ of the $i^\star$-th secret key, we sample $\boxed{\mathbf{d}_{j^\star} \leftarrow \mathrm{span}(\mathbf{B})}$ instead of $\mathbf{d}_{j^\star} \leftarrow \mathbb{Z}_p^{k+1}$.

$\mathbf{Game}_{i^\star,\ell+2}$: This game is identical to $\mathbf{Game}_{i^\star,\ell+1,1}$, except that the secret key $\mathsf{sk}_{\mathbf{L},\rho} = \{K_{0,j}, K_{1,j}, K_{2,j}\}_{j\in[\ell]}$ for the $i^\star$-th key extraction query is sampled as

$$(K_{0,j}, K_{1,j}, K_{2,j}) :=$$
$$\begin{cases} \left([(\mathbf{k} + \boxed{\alpha\mathbf{a}_2^{\|}} \| \mathbf{K}')\mathbf{L}_j^\top + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j + r_j\mathbf{a}_2^{\|}]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2\right) & \text{If } \rho(j) \notin S \\ \left([(\mathbf{k} + \boxed{\alpha\mathbf{a}_2^{\|}} \| \mathbf{K}')\mathbf{L}_j^\top + \hat{\mathbf{V}}_{\rho(j)}\mathbf{d}_j]_2,\ [\mathbf{d}_j]_2,\ [\hat{\mathbf{U}}_{\rho(j)}\mathbf{d}_j]_2\right) & \text{If } \rho(j) \in S \end{cases}$$

where $\mathbf{d}_j \leftarrow \mathrm{span}(\mathbf{B})$.

We note that $\mathbf{Game}_{i^\star,1,1}$ and $\mathbf{Game}_{i^\star}$ are identical, $\mathbf{Game}_{i^\star,j^\star,4}$ and $\mathbf{Game}_{i^\star,j^\star+1,1}$ are identical, and $\mathbf{Game}_{i^\star,\ell+2}$ and $\mathbf{Game}_{i^\star+1}$ are identical. Therefore, to complete the proof of Lemma C.1.8, it suffices to show Lemma C.1.9, C.1.10, C.1.11, and C.1.12 in the following. $\square$

Here, we recall that we denote the advantage of a PPT adversary $\mathcal{A}$ in $\mathbf{Game}_{\mathrm{xx}}$ by A_{xx}.

Lemma C.1.9. *Under the* MDDH_k *assumption on* $\mathbb{G}_2$*, we have* $|\mathsf{A}_{i^\star,j^\star,1} - \mathsf{A}_{i^\star,j^\star,2}| = \mathrm{negl}(\lambda)$ *for* $i^\star \in [q]$ *and* $j^\star \in [\ell]$.

Proof. For the sake of contradiction, we assume that $\mathcal{A}$ distinguishes $\mathbf{Game}_{i^\star,j^\star,1}$ and $\mathbf{Game}_{i^\star,j^\star,2}$ with non-negligible and show that we can construct another adversary $\mathcal{B}$ against MDDH_k with the same advantage. At the beginning of the game, $\mathcal{B}$ is given an instance $(\mathbb{G}, [\mathbf{B}]_2, [\mathbf{t}]_2)$ of MDDH_k, and proceeds as follows.

Setup. $\mathcal{B}$ first samples $(\mathbf{A}_1, \mathbf{a}_2, \mathbf{A}_3) \leftarrow \mathbb{Z}_p^{(2k+1)\times k} \times \mathbb{Z}_p^{2k+1} \times \mathbb{Z}_p^{(2k+1)\times k}$, $\mathbf{W}, \mathbf{W}_0, \mathbf{W}_1 \leftarrow \mathbb{Z}_p^{(2k+1)\times(k+1)}$, $\mathbf{k} \leftarrow \mathbb{Z}_p^{2k+1}$, and $\alpha \leftarrow \mathbb{Z}_p$. It then set $\mathsf{mpk} = ([\mathbf{A}_1^\top, \mathbf{A}_1^\top\mathbf{W}, \mathbf{A}_1^\top\mathbf{W}_0, \mathbf{A}_1^\top\mathbf{W}_1]_1, e([\mathbf{A}_1^\top]_1, [\mathbf{k}]_2))$ and gives it to $\mathcal{A}$. $\mathcal{A}$ then provides its target $(S, 1^{s_{\max}})$ to $\mathcal{B}$. $\mathcal{B}$ also samples $\mathbf{U}_s^{(2)}, \mathbf{V}_s^{(2)} \leftarrow \mathrm{span}^{k+1}(\mathbf{a}_2^{\|})$ and computes $\hat{\mathbf{V}}_s := \mathbf{W} + \mathbf{V}_s^{(2)}$ and $\hat{\mathbf{U}}_s := \mathbf{W}_0 + s\mathbf{W}_1 + \mathbf{U}_s^{(2)}$ for $s \in [n]$, where n is the upper bound on the running time of $\mathcal{A}$.

Simulating Ciphertext. When $\mathcal{A}$ asks for the challenge ciphertext with respect to messages (μ_0, μ_1), it generates E-normal ciphertext using $\mathbf{A}_1$, $\mathbf{a}_2$, $\{\hat{\mathbf{U}}_s, \hat{\mathbf{V}}_s\}_{s\in[n]}$, and $\mathbf{k}$. We note that we have $n \geq s_{\max} = \max_{s\in S}|s|$ and thus the terms $\{\hat{\mathbf{U}}_s, \hat{\mathbf{V}}_s\}_{s\in[n]}$ will suffice to simulate the ciphertext.

Simulating Keys. For the i-th key query $((\mathbf{L}, \rho), 1^{\rho_{\max}})$ made by $\mathcal{A}$, $\mathcal{B}$ proceeds as follows.

- If $i \le i^\star - 1$, it computes SF key using $\mathbf{k}$, $\mathbf{a}_2^{\parallel}$, $[\mathbf{B}]_2$, and $\{\hat{\mathbf{U}}_s, \hat{\mathbf{V}}_s\}_{s\in[n]}$. Here, $[\mathbf{B}]_2$ is used to sample $[\mathbf{d}_j]_2$ where $\mathbf{d}_j \leftarrow \mathrm{span}(\mathbf{B})$. We also note that we have $n \ge \rho_{\max} = \max_{j\in[\ell]} |\rho(j)|$ and thus the terms $\{\hat{\mathbf{U}}_s, \hat{\mathbf{V}}_s\}_{s\in[n]}$ will suffice to simulate the key.
- If $i > i^\star$, it computes E-normal key using $\mathbf{k}$, α, $\mathbf{a}_2^{\parallel}$, $[\mathbf{B}]_2$, and $\{\hat{\mathbf{U}}_s, \hat{\mathbf{V}}_s\}_{s\in[n]}$. Again, $[\mathbf{B}]_2$ is used to sample $[\mathbf{d}_j]_2$ and the terms $\{\hat{\mathbf{U}}_s, \hat{\mathbf{V}}_s\}_{s\in[n]}$ will suffice to simulate the key.
- If $i = i^\star$, it computes the secret key $\{K_{0,j}, K_{1,j}, K_{2,j}\}_{j\in[\ell]}$ as follows. The j-th component of the key $(K_{0,j}, K_{1,j}, K_{2,j})$ for $j \le j^\star - 1$ can be computed similarly to an SF key, while the j-th component for $j \ge j^\star + 1$ can be computed similarly to an E-normal key. It also computes
$$K_{0,j^\star} = [(\mathbf{k}\|\mathbf{K}')\mathbf{L}_{j^\star}^\top + \hat{\mathbf{V}}_{\rho(j^\star)}\mathbf{t}]_2, \quad K_{1,j^\star} = [\mathbf{t}]_2, \quad K_{2,j^\star} = [\hat{\mathbf{U}}_{\rho(j^\star)}\mathbf{t}]_2$$
from the challenge instance $([\mathbf{B}]_2, [\mathbf{t}]_2)$ of MDDH_k, $\hat{\mathbf{V}}_{\rho(j^\star)}$, $\hat{\mathbf{U}}_{\rho(j^\star)}$, $\mathbf{k}$, and $\mathbf{K}'$.

It is easy to see that $\mathcal{B}$ simulates $\mathbf{Game}_{i^\star,j^\star,1}$ if $\mathbf{t} \leftarrow \mathrm{span}(\mathbf{B})$ and $\mathbf{Game}_{i^\star,j^\star,2}$ if $\mathbf{t} \leftarrow \mathbb{Z}_p^{k+1}$. From this observation, Lemma C.1.9 readily follows. □

Lemma C.1.10. *For $i^\star \in [q]$ and $j^\star \in [\ell]$, we have $|\mathsf{A}_{i^\star,j^\star,2} - \mathsf{A}_{i^\star,j^\star,3}| = \mathrm{negl}(\lambda)$ unconditionally.*

Proof. We assume $\rho(j^\star) \notin S$, since otherwise $\mathbf{Game}_{i^\star,j^\star,2}$ and $\mathbf{Game}_{i^\star,j^\star,3}$ are exactly the same. We fix all randomness during the game other than $\mathbf{V}^{(2)}_{\rho(j^\star)} \leftarrow \mathrm{span}^{k+1}(\mathbf{a}_2^{\parallel})$. Let $\mathbf{b}^{\parallel}$ be a fixed non-zero vector in $\mathbb{Z}_p^{k+1}$ satisfying $\mathbf{B}^\top\mathbf{b}^{\parallel} = \mathbf{0}$. It is direct to see that $\mathbf{V}^{(2)}_{\rho(j^\star)} \leftarrow \mathrm{span}^{k+1}(\mathbf{a}_2^{\parallel})$ and $\mathbf{V}^{(2)}_{\rho(j^\star)} + v\mathbf{a}_2^{\parallel}\mathbf{b}^{\parallel\top}$ for $v \leftarrow \mathbb{Z}_p$ follow the same distribution. We then further fix $\mathbf{V}^{(2)}_{\rho(j^\star)}$ and prove that if we substitute $\mathbf{V}^{(2)}_{\rho(j^\star)}$ in $\mathbf{Game}_{i^\star,j^\star,2}$ with $\mathbf{V}^{(2)}_{\rho(j^\star)} + v\mathbf{a}_2^{\parallel}\mathbf{b}^{\parallel\top}$, the view of the adversary is the same as that in $\mathbf{Game}_{i^\star,j^\star,3}$ with the randomness other than $r_{j^\star}$ being fixed. This can be seen by the following observation:

- $\mathbf{V}^{(2)}_{\rho(j^\star)}$ is not used to generate the challenge ciphertext in both games since $\rho(j^\star) \notin S$. Therefore, even if we substitute the value with $\mathbf{V}^{(2)}_{\rho(j^\star)} + v\mathbf{a}_2^{\parallel}\mathbf{b}^{\parallel\top}$, this does not change the challenge ciphertext at all.
- We have
$$(\mathbf{V}^{(2)}_{\rho(j^\star)} + v\mathbf{a}_2^{\parallel}\mathbf{b}^{\parallel\top})\mathbf{B} = \mathbf{V}^{(2)}_{\rho(j^\star)}\mathbf{B}.$$
Therefore, the answer for the i-th key extraction query for $i \neq i^\star$ will not be changed even if we substitute $\mathbf{V}^{(2)}_{\rho(j^\star)}$ with $\mathbf{V}^{(2)}_{\rho(j^\star)} + v\mathbf{a}_2^{\parallel}\mathbf{b}^{\parallel\top}$. Because of the same reason, the j-th component in the $i^\star$-th secret key with $j \neq j^\star$ is unchanged by the substitution.

- For the $j^\star$-th components for the i-th secret key, we have

  $$(\mathbf{k}\|\mathbf{K}')\mathbf{L}_{j^\star}^\top + (\hat{\mathbf{V}}_{\rho(j^\star)} + v\mathbf{a}_2^\| \mathbf{b}^{\|^\top})\mathbf{d}_{j^\star} = (\mathbf{k}\|\mathbf{K}')\mathbf{L}_{j^\star}^\top + \hat{\mathbf{V}}_{\rho(j^\star)}\mathbf{d}_{j^\star} + r_{j^\star}\mathbf{a}_2^\|$$

  where $r_{j^\star} = v\mathbf{b}^{\|^\top}\mathbf{d}_{j^\star}$. We have $\mathbf{b}^{\|^\top}\mathbf{d}_{j^\star} \neq 0$ with probability $1 - 1/p$ since $\mathbf{d}_{j^\star} \leftarrow \mathbb{Z}_p^{k+1}$. Therefore, we have $r_{j^\star}$ is distributed uniformly at random over $\mathbb{Z}_p$ since so is v. Here, we use the fact that v is not used elsewhere in the game. It is readily seen that $(K_{0,j^\star}, K_{1,j^\star}, K_{2,j^\star})$ is distributed as in $\mathbf{Game}_{i^\star,j^\star,3}$.

This completes the proof of Lemma C.1.10. □

Lemma C.1.11. *Under the* MDDH_k *assumption on* $\mathbb{G}_2$*, we have* $|\mathsf{A}_{i^\star,j^\star,3} - \mathsf{A}_{i^\star,j^\star,4}| = \mathrm{negl}(\lambda)$ *for* $i^\star \in [q]$ *and* $j^\star \in [\ell]$.

Proof. The proof is completely analogous to that of Lemma C.1.9 except that we compute the $j^\star$-th component of the $i^\star$-th key is computed as

$$K_{0,j^\star} = [(\mathbf{k}\|\mathbf{K}')\mathbf{L}_{j^\star}^\top + \hat{\mathbf{V}}_{\rho(j^\star)}\mathbf{t} + r_{j^\star}\mathbf{a}_2^\|]_2, \quad K_{1,j^\star} = [\mathbf{t}]_2, \quad K_{2,j^\star} = [\hat{\mathbf{U}}_{\rho(j^\star)}\mathbf{t}]_2.$$

□

Lemma C.1.12. *For* $i^\star \in [q]$ *and* $j^\star \in [\ell]$*, we have* $|\mathsf{A}_{i^\star,\ell+1,1} - \mathsf{A}_{i^\star,\ell+2}| = \mathrm{negl}(\lambda)$ *unconditionally.*

Proof. Let us fix all the randomness used in the games except for that used for generating the $i^\star$-th secret key. Let $(\mathbf{L} \in \mathbb{Z}_p^{\ell\times m}, \rho)$ be the span program associated to the $i^\star$-th secret key. By the definition of $\mathbf{Game}_{i^\star,\ell+1}$ and $\mathbf{Game}_{i^\star,\ell+2}$, it suffices show that the following distributions are the same:

$$\{(\mathbf{0}\|\mathbf{K}')\mathbf{L}_j^\top + r_j\delta_j\mathbf{a}_2^\|\}_{j\in[\ell]} \approx \{(\alpha\mathbf{a}_2^\|\|\mathbf{K}')\mathbf{L}_j^\top + r_j\delta_j\mathbf{a}_2^\|\}_{j\in[\ell]} \qquad \text{(C.1)}$$

where $\mathbf{K}' \leftarrow \mathbb{Z}_p^{(2k+1)\times(m-1)}$, $r_j \leftarrow \mathbb{Z}_p$, δ_j is defined to be $\delta_j = 0$ if $\rho(j) \in S$ and $\delta_j = 1$ if $\rho(j) \notin S$ for the attribute S associated to the challenge ciphertext. To see this, we first observe that by Lemma C.1.3 and from the fact that S does not satisfy $(\mathbf{L}, \rho)$, the following distributions are the same:

$$\{(0\|\mathbf{k}')\mathbf{L}_j^\top + r_j\delta_j\}_{j\in[\ell]} \approx \{(1\|\mathbf{k}')\mathbf{L}_j^\top + r_j\delta_j\}_{j\in[\ell]}$$

where $\mathbf{k}'$ is a row vector sampled as $\mathbf{k}' \leftarrow \mathbb{Z}_p^{m-1}$. By multiplying $\mathbf{a}_2^\|$ from the left and adding $(\mathbf{0}\|\mathbf{K}'')\mathbf{L}_j$ for both distributions with $\mathbf{K}'' \leftarrow \mathbb{Z}_p^{(2k+1)\times(m-1)}$, we have that the following distributions are the same:

$$\{(\mathbf{0}\|\mathbf{a}_2^{\|}\mathbf{k}' + \mathbf{K}'')\mathbf{L}_j^\top + r_j\delta_j\mathbf{a}_2^{\|}\}_{j\in[\ell]} \approx \{(\alpha\mathbf{a}_2^{\|}\|\mathbf{a}_2^{\|}\mathbf{k}' + \mathbf{K}'')\mathbf{L}_j^\top + r_j\delta_j\mathbf{a}_2^{\|}\}_{j\in[\ell]}.$$

By setting $\mathbf{K}' = \mathbf{a}_2^{\|}\mathbf{k}' + \mathbf{K}''$, we can see that the left and the right distributions in the above equation correspond to those of Eq. (C.1). This completes the proof of Lemma C.1.12. $\square$

C.1.4 The Construction of Ingredient CP-ABE

Here, we provide an ABE scheme for R^{MUCP}, denoted by cpABE. The construction is essentially the same as the unbounded CP-ABE given in [Chen *et al.* (2018)] with the modification that we allow unbounded multi-use of the same attribute in an MSP.

Our construction cpABE for relation R^{MUCP} is defined below.

$\mathsf{Setup}(1^\lambda)$**:** On input 1^λ, sample

$$\mathbf{A}_1 \leftarrow \mathbb{Z}_p^{3k\times k}, \mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}, \mathbf{W}, \mathbf{W}_0, \mathbf{W}_1, \mathbf{U}_0 \leftarrow \mathbb{Z}_p^{3k\times(k+1)}, \mathbf{k} \leftarrow \mathbb{Z}_p^{3k}$$

and output

$$\mathsf{mpk} := \big([\mathbf{A}_1^\top, \mathbf{A}_1^\top\mathbf{W}, \mathbf{A}_1^\top\mathbf{W}_0, \mathbf{A}_1^\top\mathbf{W}_1, \mathbf{A}_1^\top\mathbf{U}_0]_1, e([\mathbf{A}_1^\top]_1, [\mathbf{k}]_2)\big) \in \mathbb{G}_1^{k\times 3k} \times (\mathbb{G}_1^{k\times(k+1)})^4 \times \mathbb{G}_T^k$$

and

$$\mathsf{msk} := (\mathbf{k}, \mathbf{B}, \mathbf{W}, \mathbf{W}_0, \mathbf{W}_1, \mathbf{U}_0).$$

$\mathsf{Enc}(\mathsf{mpk}, ((\mathbf{L}, \rho), 1^{\rho_{\max}}), \mu)$**:** On input a monotone span program $(\mathbf{L}, \rho)$ such that $\mathbf{L} \in \mathbb{Z}_p^{\ell\times m}$, and $\mu \in \mathbb{G}_T$, pick $\mathbf{c}, \mathbf{c}_j \leftarrow \mathrm{span}(\mathbf{A}_1)$ for all $j \in [\ell]$, sample $\mathbf{U} \leftarrow \mathbb{Z}_p^{(m-1)\times(k+1)}$ and output

$$\mathsf{ct}_{(\mathbf{L},\rho)} := \left(\begin{array}{c} C_0 := [\mathbf{c}^\top]_1,\ C := e([\mathbf{c}^\top]_1, [\mathbf{k}]_2)\cdot\mu, \\ \left\{C_{1,j} := [\mathbf{L}_j \left(\begin{smallmatrix}\mathbf{c}^\top\mathbf{U}_0 \\ \mathbf{U}\end{smallmatrix}\right) + \mathbf{c}_j^\top\mathbf{W}]_1,\ C_{2,j} := [\mathbf{c}_j^\top]_1,\ C_{3,j} := [\mathbf{c}_j^\top(\mathbf{W}_0 + \rho(j)\mathbf{W}_1)]_1\right\}_{j\in[\ell]} \end{array}\right),$$

where $\mathbf{L}_j$ is the j-th row of $\mathbf{L}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{mpk}, (S, 1^{s_{\max}}))$: On input an attribute set $S = \{s_1, \ldots, s_\ell\} \subset \mathbb{Z}$, pick $\mathbf{d}, \mathbf{d}_s \leftarrow \mathrm{span}(\mathbf{B})$ for all $s \in S$ and output

$$\mathsf{sk}_S := \left(\begin{array}{c} K_0 := [\mathbf{k} + \mathbf{U}_0\mathbf{d}]_2, K_1 := [\mathbf{d}]_2, \\ \{K_{2,s} := [\mathbf{W}\mathbf{d} + (\mathbf{W}_0 + s \cdot \mathbf{W}_1)\mathbf{d}_s]_2,\ K_{3,s} := [\mathbf{d}_s]_2\}_{s \in S} \end{array} \right).$$

$\mathsf{Dec}(\mathsf{mpk}, \mathsf{ct}, ((\mathbf{L}, \rho), 1^{\rho_{\max}}), \mathsf{sk}_{(\mathbf{L},\rho)}, (\mathsf{S}, 1^{\mathsf{s}_{\max}}))$: Since S satisfies $(\mathbf{L}, \rho)$, one can compute $\{\omega_j\}_{j \in [\ell]}$ such that

$$\sum_{j:\rho(j) \in S} \omega_j \mathbf{L}_j = (1, 0, \ldots, 0).$$

Then, compute

$$K = e(C_0, K_0) / \prod_{j:\rho(j) \in S} \left(e(C_{1,j}, K_1) \cdot e(C_{2,j}, K_{2,\rho(j)})^{-1} \cdot e(C_{3,j}, K_{3,\rho(j)}) \right)^{\omega_j}$$

and retrieve the message by C/K.

Correctness. For all j such that $\rho(j) \in S$, we have

$$\begin{aligned} & e(C_{1,j}, K_1) \cdot e(C_{2,j}, K_{2,\rho(j)})^{-1} \cdot e(C_{3,j}, K_{3,\rho(j)}) \\ = \ & e([\mathbf{L}_j \left(\begin{smallmatrix} \mathbf{c}^\top \mathbf{U}_0 \\ \mathbf{U} \end{smallmatrix} \right) + \mathbf{c}_j^\top \mathbf{W}]_1, [\mathbf{d}]_2) \cdot e([\mathbf{c}_j^\top]_1, [\mathbf{W}\mathbf{d} + (\mathbf{W}_0 + \rho(j) \cdot \mathbf{W}_1)\mathbf{d}_{\rho(j)}]_2)^{-1} \\ & \cdot e([\mathbf{c}_j^\top (\mathbf{W}_0 + \rho(j) \cdot \mathbf{W}_1)]_1, [\mathbf{d}_{\rho(j)}]_2) \\ = \ & [\mathbf{L}_j \left(\begin{smallmatrix} \mathbf{c}^\top \mathbf{U}_0 \mathbf{d} \\ \mathbf{U}\mathbf{d} \end{smallmatrix} \right)]_T \end{aligned}$$

for all $j \in [\ell]$. The correctness readily follows from the following equation.

$$\begin{aligned} K = e(C_0, K_0) / \prod_{j:\rho(j) \in S} [\mathbf{L}_j \left(\begin{smallmatrix} \mathbf{c}^\top \mathbf{U}_0 \mathbf{d} \\ \mathbf{U}\mathbf{d} \end{smallmatrix} \right)]_T^{\omega_j} &= [\mathbf{c}^\top \mathbf{k}]_T \cdot [\mathbf{c}^\top \mathbf{U}_0 \mathbf{d}]_T / [\sum_{j:\rho(j) \in S} \omega_j \mathbf{L}_j \left(\begin{smallmatrix} \mathbf{c}^\top \mathbf{U}_0 \mathbf{d} \\ \mathbf{U}\mathbf{d} \end{smallmatrix} \right)]_T \\ &= [\mathbf{c}^\top \mathbf{k}]_T \cdot [\mathbf{c}^\top \mathbf{U}_0 \mathbf{d}]_T / [\mathbf{c}^\top \mathbf{U}_0 \mathbf{d}]_T = [\mathbf{c}^\top \mathbf{k}]_T. \end{aligned}$$

C.1.5 Security Proof

Here, we prove selective* (please see Section 4.7.3) security of the CP-ABE scheme in Appendix C.1.4. To prove the security, we first recall the prime-order bilinear entropy expansion lemma for CP-ABE from [Chen *et al.* (2018)].

Lemma C.1.13 (Lemma 14 from [Chen *et al.* (2018)] with $\ell_1 = \ell_2 = \ell_3 = k$, $\ell_W = k+1$)**.** *Pick basis* $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3) \leftarrow (\mathbb{Z}_p^{3k\times k})^3$ *and define its dual* $(\mathbf{A}_1^{\|}, \mathbf{A}_2^{\|}, \mathbf{A}_3^{\|})$ *such that* $\mathbf{A}_i^\top \mathbf{A}_j = \mathbf{I}$ *if* $i = j$ *and* $\mathbf{A}_i^\top \mathbf{A}_j = \mathbf{0}$ *otherwise. With* $\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$ *and for any polynomially bounded* $n \in \mathbb{N}$*, we have*

$$\begin{Bmatrix} \mathsf{aux}: & [\mathbf{A}_1^\top]_1, [\mathbf{A}_1^\top\mathbf{W}]_1, [\mathbf{A}_1^\top\mathbf{W}_0]_1, [\mathbf{A}_1^\top\mathbf{W}_1]_1 \\ \mathsf{ct}: & [\mathbf{c}^\top]_1, \{[\mathbf{c}_s^\top\mathbf{W}]_1, [\mathbf{c}_s]_1, [\mathbf{c}_s^\top(\mathbf{W}_0 + s\cdot\mathbf{W}_1)]_1\}_{s\in[n]} \\ \mathsf{sk}: & \{[\mathbf{D}]_2, [\mathbf{W}\mathbf{D} + (\mathbf{W}_0 + s\cdot\mathbf{W}_1)\mathbf{D}_s]_2, [\mathbf{D}_s]_2\}_{s\in[n]} \end{Bmatrix}$$
$$\overset{c}{\approx}\begin{Bmatrix} \mathsf{aux}: & [\mathbf{A}_1^\top]_1, [\mathbf{A}_1^\top\mathbf{W}]_1, [\mathbf{A}_1^\top\mathbf{W}_0]_1, [\mathbf{A}_1^\top\mathbf{W}_1]_1 \\ \mathsf{ct}: & [\boxed{\mathbf{c}}^\top]_1, \{[\boxed{\mathbf{c}_s}^\top(\mathbf{W} + \boxed{\mathbf{V}_s^{(2)}})]_1, [\boxed{\mathbf{c}_s}]_1, [\boxed{\mathbf{c}_s}^\top(\mathbf{W}_0 + s\cdot\mathbf{W}_1 + \boxed{\mathbf{U}_s^{(2)}})]_1\}_{s\in[n]} \\ \mathsf{sk}: & \{[\mathbf{D}]_2, [(\mathbf{W} + \boxed{\mathbf{V}_s^{(2)}})\mathbf{D} + (\mathbf{W}_0 + s\cdot\mathbf{W}_1 + \boxed{\mathbf{U}_s^{(2)}})\mathbf{D}_s]_2, [\mathbf{D}_s]_2\}_{s\in[n]} \end{Bmatrix},$$

where $\mathbf{W}, \mathbf{W}_0, \mathbf{W}_1 \leftarrow \mathbb{Z}_p^{3k\times(k+1)}, \mathbf{V}_s^{(2)}, \mathbf{U}_s^{(2)} \leftarrow \mathrm{span}^{k+1}(\mathbf{A}_2^{\|}), \mathbf{D}, \mathbf{D}_s \leftarrow \mathrm{span}^{(k+1)}(\mathbf{B})$*, and* $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1)$ *in the left distribution while* $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{A}_2)$ *in the right distribution.*

Note that in [Chen *et al.* (2018)], $\mathbf{D}$ and $\mathbf{D}_s$ are sampled from $\mathbb{Z}_p^{(k+1)\times(k+1)}$ while we sample them from $\mathrm{span}^{(k+1)}(\mathbf{B})$. The distributions in the lemma are still computationally indistinguishable even with this change due to the MDDH_k assumption.

We also prove the following lemma, which will be used in the core part of our security proof.

Lemma C.1.14. *For any set of integers* S *and span program* $(\mathbf{L} \in \mathbb{Z}_p^{\ell\times m}, \rho)$ *such that* S *does not satisfy* $(\mathbf{L}, \rho)$*, we have that the following distributions are computationally indistinguishable under the DLIN assumption.*

$$\left\{\mathsf{ct} := \left([\mathsf{c}]_1, \left\{[\mathbf{L}_\mathsf{j}\left(\begin{smallmatrix} cu_0 \\ \mathbf{u}\end{smallmatrix}\right) + \mathsf{c}_\mathsf{j}\mathsf{v}_{\rho(\mathsf{j})}]_1, [\mathsf{c}_\mathsf{j}]_1\right\}_{\mathsf{j}\in[\ell]}\right),\ \mathsf{sk} := \left([\mathsf{u}_0]_2, \{[\mathsf{v}_\mathsf{s}]_2\}_{\mathsf{s}\in\mathsf{S}}\right)\right\}$$
$$\approx_c \left\{\mathsf{ct} := \left([\mathsf{c}]_1, \left\{[\mathbf{L}_\mathsf{j}\left(\begin{smallmatrix} cu_0 \\ \mathbf{u}\end{smallmatrix}\right) + \mathsf{c}_\mathsf{j}\mathsf{v}_{\rho(\mathsf{j})}]_1, [\mathsf{c}_\mathsf{j}]_1\right\}_{\mathsf{j}\in[\ell]}\right),\ \mathsf{sk} := \left([\mathsf{u}_0 + \boxed{\alpha}]_2, \{[\mathsf{v}_\mathsf{s}]_2\}_{\mathsf{s}\in\mathsf{S}}\right)\right\}$$

where $c, \alpha, u_0 \leftarrow \mathbb{Z}_p$*,* $\mathbf{u} \leftarrow \mathbb{Z}_p^{m-1}$*,* $c_j \leftarrow \mathbb{Z}_p$ *for* $j \in [\ell]$*, and* $v_s \leftarrow \mathbb{Z}_p$ *for* $s \in S \cup \{\rho(j) | j \in [\ell]\}$.

Proof. We construct an attacker $\mathcal{B}$ against the DLIN assumption assuming the distinguisher $\mathcal{A}$ against the distributions. Given the problem instance $([x_1]_1, [x_2]_1, [x_1y_1]_1, [x_2y_2]_1, [\Phi]_2)$ of the DLIN assumption, $\mathcal{B}$ proceeds as follows. Let us define $T := S \cup \{\rho(j) | j \in [\ell]\}$. $\mathcal{B}$ samples $\tilde{v}_s \leftarrow \mathbb{Z}_p$ for $s \in T$ and implicitly sets

$$u_0 := y_1 + y_2, \quad v_s := \begin{cases} \tilde{v}_s & \text{for } s \in S \\ \tilde{v}_s - x_1/x_2 & \text{for } s \in T \backslash S \end{cases}.$$

It can be seen that these components are distributed uniformly at random over $\mathbb{Z}_p$ as desired. $\mathcal{B}$ sets sk as

$$\mathsf{sk} = \left([\Phi]_2, \{[\tilde{v}_s]_2\}_{s \in S}\right).$$

It is easy to see that it simulates the left distribution if $[\Phi]_2 = [y_1 + y_2]_2$ and the right otherwise. To compute ct, $\mathcal{B}$ first computes a vector $\left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right)$ satisfying $\mathbf{L}_j \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) = 0$ for all j such that $\rho(j) \in S$. Such a vector exists and can be computed efficiently because S does not satisfy $(\mathbf{L}, \rho)$ (See for example Proposition 1 in [Goyal *et al.* (2006)]). $\mathcal{B}$ then picks $\tilde{\mathbf{u}} \leftarrow \mathbb{Z}_p^{m-1}$ and implicitly sets

$$c = x_1, \quad \mathbf{u} = \tilde{\mathbf{u}} + cu_0\tilde{\mathbf{t}}, \quad c_j = \begin{cases} \tilde{c}_j & \text{if } \rho(j) \in S \\ \mathbf{L}_j \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) x_2y_2 + \tilde{c}_j x_2 & \text{if } \rho(j) \notin S \end{cases}$$

We observe that these components are distributed uniformly at random over $\mathbb{Z}_p$ as desired. We then check that each component in ct is efficiently computable. First, we have $[c]_1$ and $[c_j]_1$ for $j \in [\ell]$ are computable from $[x_1]_1$, $[x_2]_1$ and $[x_2y_2]_1$. We then observe that $[\mathbf{L}_j \left(\begin{smallmatrix} cu_0 \\ \mathbf{u} \end{smallmatrix}\right) + c_j v_{\rho(j)}]_1$ can be computed for j such that $\rho(j) \in S$ since we have

$$\mathbf{L}_j \left(\begin{smallmatrix} cu_0 \\ \mathbf{u} \end{smallmatrix}\right) + c_j v_{\rho(j)} = \mathbf{L}_j \left(cu_0 \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) + \left(\begin{smallmatrix} 0 \\ \tilde{\mathbf{u}} \end{smallmatrix}\right)\right) + \tilde{c}_j \tilde{v}_{\rho(j)} = \mathbf{L}_j \left(\begin{smallmatrix} 0 \\ \tilde{\mathbf{u}} \end{smallmatrix}\right) + \tilde{c}_j \tilde{v}_{\rho(j)},$$

where all components are known to $\mathcal{B}$. We then observe that for j such that $\rho(j) \notin S$, it holds

$$\begin{aligned} \mathbf{L}_j \left(\begin{smallmatrix} cu_0 \\ \mathbf{u} \end{smallmatrix}\right) + c_j v_{\rho(j)} &= \mathbf{L}_j \left(cu_0 \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) + \left(\begin{smallmatrix} 0 \\ \tilde{\mathbf{u}} \end{smallmatrix}\right)\right) + \left(-x_1/x_2 + \tilde{v}_{\rho(j)}\right) \cdot c_j \\ &= \mathbf{L}_j \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) \cdot x_1(y_1 + y_2) + \mathbf{L}_j \left(\begin{smallmatrix} 0 \\ \tilde{\mathbf{u}} \end{smallmatrix}\right) + \tilde{v}_{\rho(j)} \cdot c_j - (x_1/x_2)c_j \\ &= \mathbf{L}_j \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) \cdot x_1(y_1 + y_2) + \mathbf{L}_j \left(\begin{smallmatrix} 0 \\ \tilde{\mathbf{u}} \end{smallmatrix}\right) + \tilde{v}_{\rho(j)} \cdot c_j - \mathbf{L}_j \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) x_1y_2 - \tilde{c}_j \cdot x_1 \\ &= \mathbf{L}_j \left(\begin{smallmatrix} 1 \\ \tilde{\mathbf{t}} \end{smallmatrix}\right) x_1y_1 + \mathbf{L}_j \left(\begin{smallmatrix} 0 \\ \tilde{\mathbf{u}} \end{smallmatrix}\right) + \tilde{v}_{\rho(j)} \cdot c_j - \tilde{c}_j \cdot x_1. \end{aligned}$$

Therefore, we can compute $[\mathbf{L}_j \binom{cu_0}{\mathbf{u}} + c_j v_{\rho(j)}]_1$ from $[x_1 y_1]_1$, $[c_j]_1$, and $[x_1]_1$. Note that $x_1 y_2$, which is problematic when simulating the term, cancels out in the above computation. This completes the proof of the lemma. □

We are now ready to state and prove the main theorem. The proof is very similar to that of [Chen *et al.* (2018)], but since certain information theoretic step in [Chen *et al.* (2018)] does not work in the multi-use setting, we replace it with computational argument using Lemma C.1.14.

Theorem C.1.15. *The ABE scheme for relation R^{CPMU} (i.e., multi-use key-policy unbounded ABE with polynomial valued attributes) in Appendix C.1.4 satisfies selective* security under the* DLIN *assumption.*

Proof. To prove the theorem, we define various forms of ciphertext (of message μ for span program $(\mathbf{L}, \rho)$).

- Normal: Generated by Enc; in particular, $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1)$.
- E-normal: Same as a normal ciphertext except that $\mathbf{c}, \mathbf{c}_s \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{A}_2)$ and we use the substitution:

$$\mathbf{W} \mapsto \widehat{\mathbf{V}}_{\rho(j)} := \mathbf{W} + \mathbf{V}^{(2)}_{\rho(j)} \text{ in } j\text{-th component and}$$
$$\mathbf{W}_0 + \rho(j) \cdot \mathbf{W}_1 \mapsto \widehat{\mathbf{U}}_{\rho(j)} := \mathbf{W}_0 + \rho(j) \cdot \mathbf{W}_1 + \mathbf{U}^{(2)}_{\rho(j)}$$

  where $\mathbf{U}^{(2)}_s, \mathbf{V}^{(2)}_s \leftarrow \mathrm{span}^{k+1}(\mathbf{A}_2^{\|})$. Concretely, an E-normal ciphertext is of the form

$$\mathsf{ct}_{(\mathbf{L},\rho)} := \left([\mathbf{c}^\top]_1, \{[\mathbf{L}_\mathsf{j} \binom{\mathbf{c}^\top \mathbf{U}_0}{\mathbf{U}} + \mathbf{c}_\mathsf{j}^\top \boxed{\widehat{\mathbf{V}}_{\rho(j)}}]_1, [\mathbf{c}_\mathsf{j}^\top]_1, [\mathbf{c}_\mathsf{j}^\top \boxed{\widehat{\mathbf{U}}_{\rho(j)}}]_1\}_{\mathsf{j}\in[\mathsf{n}]}, \mathsf{e}([\mathbf{c}^\top]_1, [\mathbf{k}]_2) \cdot \mu \right)$$

  where $\mathbf{U} \leftarrow \mathbb{Z}_p^{(m-1)\times(k+1)}$.

Then we pick $\mathbf{k}^{(2)} \leftarrow \mathrm{span}(\mathbf{A}_2^{\|})$ and define various forms of key (for attribute S):

- Normal: Generated by KeyGen.
- E-normal: Same as a Normal key except that we use the same substitution as in Eq. (C.2). Concretely, an E-normal key is of the form

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0 \mathbf{d}]_2, [\mathbf{d}]_2, \{[\boxed{\widehat{\mathbf{V}}_s} \mathbf{d} + \boxed{\widehat{\mathbf{U}}_s} \mathbf{d}_s]_2\ [\mathbf{d}_s]_2\}_{s\in S} \right) \text{ where } \mathbf{d}, \mathbf{d}_s \leftarrow \mathrm{span}(\mathbf{B}).$$

- P-normal: Sample $\mathbf{d}, \mathbf{d}_s \leftarrow \mathbb{Z}_p^{k+1}$ in an E-normal key. Concretely, a P-normal key is of the form

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0 \mathbf{d}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s \mathbf{d} + \widehat{\mathbf{U}}_s \mathbf{d}_s]_2\ [\mathbf{d}_s]_2\}_{s\in S} \right) \text{ where } \boxed{\mathbf{d}, \mathbf{d}_s \leftarrow \mathbb{Z}_p^{k+1}}.$$

- P-SF: Replace $\mathbf{k}$ with $\mathbf{k}+\mathbf{k}^{(2)}$ in a P-normal key. Concretely, a P-SF key is of the form

$$\mathsf{sk}_S := \Big([\mathbf{k}+\boxed{\mathbf{k}^{(2)}}+\mathbf{U}_0\mathbf{d}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s\mathbf{d}+\widehat{\mathbf{U}}_s\mathbf{d}_s]_2\ [\mathbf{d}_s]_2\}_{s\in S}\Big) \text{ where } \mathbf{d},\mathbf{d}_s \leftarrow \mathbb{Z}_p^{k+1}.$$

- SF: Sample $\mathbf{d},\mathbf{d}_s \leftarrow \mathrm{span}(\mathbf{B})$ in a P-SF key. Concretely, a SF key is of the form

$$\mathsf{sk}_S := \Big([\mathbf{k}+\mathbf{k}^{(2)}+\mathbf{U}_0\mathbf{d}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s\mathbf{d}+\widehat{\mathbf{U}}_s\mathbf{d}_s]_2\ [\mathbf{d}_s]_2\}_{s\in S}\Big) \text{ where } \boxed{\mathbf{d},\mathbf{d}_s \leftarrow \mathrm{span}(\mathbf{B})}.$$

Let us fix a PPT adversary $\mathcal{A}$ and let the number of key generation queries made by an adversary be q. We define the following sequence of games to prove the security. We use exactly the same sequence of games as [Chen *et al.* (2018)]. The proof is also similar to [Chen *et al.* (2018)], except that we need to modify one particular step in their proof.

$\mathbf{Game}_0$**:** This is the real security game for semi-adaptive security where all ciphertexts and keys are normal.

$\mathbf{Game}_{0'}$: In this game, we change the challenge ciphertext and all keys to be E-normal ones. We can show $\mathbf{Game}_{0'} \approx_c \mathbf{Game}_0$ by using the bilinear expansion lemma for CP-ABE (Lemma C.1.13) in a similar manner to the proof of Lemma C.1.6.

$\mathbf{Game}_{i^\star}$**:** In this game, the first $i^\star-1$ secret keys given to the adversary are SF, while rest of the secret keys are E-normal. It is easy to see that $\mathbf{Game}_1$ is equivalent to $\mathbf{Game}_{0'}$. To show $\mathbf{Game}_{i^\star} \approx_c \mathbf{Game}_{i^\star+1}$, we will require another sequence of sub-games.

$\mathbf{Game}_{i^\star,1}$**:** Identical to $\mathbf{Game}_{i^\star}$ except that the $i^\star$-th key is P-normal. By a straightforward reduction to the MDDH_k assumption, one can show $\mathbf{Game}_{i^\star} \approx_c \mathbf{Game}_{i^\star,1}$.

$\mathbf{Game}_{i^\star,2}$**:** Identical to $\mathbf{Game}_{i^\star}$ except that the $i^\star$-th key is P-SF. To show $\mathbf{Game}_{i^\star,1} \approx_c \mathbf{Game}_{i^\star,2}$, we need some more work. We note that this is the only step that the proof in [Chen *et al.* (2018)] does not work in our multi-use setting. We will introduce another sequence of games in order to prove this.

$\mathbf{Game}_{i^\star,3}$**:** Identical to $\mathbf{Game}_{i^\star}$ except that the $i^\star$-th key is SF. We can show $\mathbf{Game}_{i^\star,2} \approx_c \mathbf{Game}_{i^\star,3}$ by a straightforward reduction to the MDDH_k assumption, similarly to the case of $\mathbf{Game}_{i^\star} \approx_c \mathbf{Game}_{i^\star,1}$. Note that $\mathbf{Game}_{i^\star,3}$ and $\mathbf{Game}_{i^\star+1}$ are equivalent.

Game$_{\text{Final}}$**:** This is the same as **Game**$_{q+1}$ except that the challenge ciphertext is a E-normal one for a random message in $\mathbb{G}_T$. By a similar proof to Lemma C.1.7, we can prove $\mathbf{Game}_{q+1} \approx_c \mathbf{Game}_{\text{Final}}$. Note that the advantage of $\mathcal{A}$ in this game is 0.

From the above discussion, it suffices to show that $\mathbf{Game}_{i^\star,1}$ and $\mathbf{Game}_{i^\star,2}$ are indistinguishable to complete the proof of Theorem C.1.15. In [Chen *et al.* (2018)], these games are shown to be *statistically* indistinguishable. However, since the statistical argument given in [Chen *et al.* (2018)] does not work in the multi-use setting, we replace it with the *computational* argument using the DLIN assumption. The idea of using computational argument instead of statistical argument to make a secret key semi-functional is taken from previous works [Lewko and Waters (2012); Attrapadung (2014, 2016)]. Note that this is the only step where our proof doe not work for the case of adaptive security. In order to prove the indistinguishability of $\mathbf{Game}_{i^\star,1}$ and $\mathbf{Game}_{i^\star,2}$, we further introduce following sequence of games.

$\mathbf{Game}_{i^\star,1,0}$: This is the same as $\mathbf{Game}_{i^\star,1}$.

$\mathbf{Game}_{i^\star,1,1}$: In this game, we change the form of the challenge ciphertext as follows. Let us pick $\mathbf{c}, \mathbf{c}_j \leftarrow \mathrm{span}(\mathbf{A}_1)$, $c \leftarrow \mathbb{Z}_p$, $\mathbf{a}_2, \mathbf{a}_{2,j} \leftarrow \mathrm{span}(\mathbf{A}_2)$ for $j \in [\ell]$. The challenge ciphertext is computed as follows:

$$\mathsf{ct}_{(\mathbf{L},\rho)} := \left(\begin{array}{c} C_0 = [\mathbf{c}^\top + c \cdot \mathbf{a}_2^\top]_1, \\ C = e([\mathbf{c}^\top + c \cdot \mathbf{a}_2^\top]_1, [\mathbf{k}]_2) \cdot \mu_\beta \end{array}, \left\{ \begin{array}{c} C_{1,j} = [\mathbf{C}_{1,j}]_1 \\ C_{2,j} = [\mathbf{c}_j^\top + \mathbf{a}_{2,j}^\top]_1, \\ C_{3,j} = [(\mathbf{c}_j^\top + \mathbf{a}_{2,j}^\top)\widehat{\mathbf{U}}_{\rho(j)}]_1 \end{array} \right\}_{\mathsf{j}\in[\ell]} \right)$$

where

$$\mathbf{C}_{1,j} = \mathbf{L}_j \begin{pmatrix} (\mathbf{c}^\top + c\mathbf{a}_2^\top)\mathbf{U}_0 \\ \mathbf{U} \end{pmatrix} + (\mathbf{c}_j + \mathbf{a}_{2,j})^\top \widehat{\mathbf{V}}_{\rho(j)}.$$

$\mathbf{Game}_{i^\star,1,2}$**:** In this game, the challenge ciphertext and the $i^\star$-th key are changed. Let $\mathbf{a}_2^{\|} \leftarrow \mathrm{span}(\mathbf{A}_2^{\|})$. Then, $i^\star$-th secret key is sampled as follows:

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0\mathbf{d} + \boxed{u_0(\mathbf{b}^{\|\top}\mathbf{d})\mathbf{a}_2^{\|}}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s\mathbf{d} + \widehat{\mathbf{U}}_s\mathbf{d}_s + \boxed{v_s(\mathbf{b}^{\|\top}\mathbf{d})\mathbf{a}_2^{\|}}]_2\ [\mathbf{d}_s]_2\}_{s\in S} \right)$$

where $\mathbf{b}^{\|}$ is some fixed vector such that $\mathbf{B}\mathbf{b}^{\|} = \mathbf{0}$, $\mathbf{d}, \mathbf{d}_s \leftarrow \mathbb{Z}_p^{k+1}$, and $u_0, v_s \leftarrow \mathbb{Z}_p$

for $s \in S$. We also change the ciphertext component $[\mathbf{C}_{1,j}]_1$ as

$$\mathbf{C}_{1,j} = \mathbf{L}_j \left(\begin{smallmatrix} (\mathbf{c}^\top + c\mathbf{a}_2^\top)\mathbf{U}_0 \\ \mathbf{U} \end{smallmatrix} \right) + \boxed{\mathbf{a}_2^\top \mathbf{a}_2^{\|} \cdot \mathbf{L}_j \left(\begin{smallmatrix} cu_0 \mathbf{b}^{\|^\top} \\ \mathbf{0} \end{smallmatrix} \right)} + (\mathbf{c}_j + \mathbf{a}_{2,j})^\top \widehat{\mathbf{V}}_{\rho(j)} + \boxed{v_{\rho(j)} \mathbf{a}_{2,j}^\top \mathbf{a}_2^{\|} \mathbf{b}^{\|^\top}},$$

for $j \in [\ell]$.

Game$_{i^\star,1,3}$: In this game, we further change how we sample $\mathbf{a}_{2,j}$ and the ciphertext component $\mathbf{C}_{1,j}$. Namely, we sample $\boxed{c_j \leftarrow \mathbb{Z}_p}$ and $\mathbf{a}_{2,j}$ for $j \in [\ell]$ as $\boxed{\mathbf{a}_{2,j} \leftarrow \mathrm{span}(\mathbf{A}_2) \text{ conditioned on } \mathbf{a}_{2,j}^\top \mathbf{a}_2^{\|} = (\mathbf{a}_2^\top \mathbf{a}_2^{\|}) c_j}$. Furthermore, we sample $\mathbf{C}_{1,j}$ as

$$\mathbf{C}_{1,j} = \mathbf{L}_j \left(\begin{smallmatrix} (\mathbf{c}^\top + c\mathbf{a}_2^\top)\mathbf{U}_0 \\ \mathbf{U} \end{smallmatrix} \right) + (\mathbf{c}_j + \mathbf{a}_{2,j})^\top \widehat{\mathbf{V}}_{\rho(j)} + \boxed{\mathbf{a}_2^\top \mathbf{a}_2^{\|} \cdot \left(\mathbf{L}_j \left(\begin{smallmatrix} cu_0 \\ \mathbf{u} \end{smallmatrix} \right) + c_j v_{\rho(j)} \right) \cdot \mathbf{b}^{\|^\top}},$$

where $\boxed{\mathbf{u} \leftarrow \mathbb{Z}_p^{m-1}}$.

Game$_{i^\star,1,4}$: In this game, we further change the $i^\star$-th secret key to be

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0 \mathbf{d} + \boxed{k\mathbf{a}_2^{\|}} + u_0(\mathbf{b}^{\|^\top} \mathbf{d})\mathbf{a}_2^{\|}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s \mathbf{d} + \widehat{\mathbf{U}}_s \mathbf{d}_s + v_s(\mathbf{b}^{\|^\top} \mathbf{d})\mathbf{a}_2^{\|}]_2 \; [\mathbf{d}_s]_2\}_{s \in S} \right).$$

Game$_{i^\star,1,5}$: In this game, we revert the challenge ciphertext to be sampled as in **Game$_{i^\star,1,0}$** (namely, it is E-normal ciphertext) and change the $i^\star$-th secret key as follows:

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0 \mathbf{d} + k\mathbf{a}_2^{\|}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s \mathbf{d} + \widehat{\mathbf{U}}_s \mathbf{d}_s]_2 \; [\mathbf{d}_s]_2\}_{s \in S} \right),$$

where $k \leftarrow \mathbb{Z}_p$ and $\mathbf{a}_2^{\|} \leftarrow \mathrm{span}(\mathbf{A}_2)$.

Game$_{i^\star,1,6}$: In this game, we change the $i^\star$-th secret key as follows:

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0 \mathbf{d} + k\mathbf{a}_2^{\|} + \boxed{\mathbf{k}^{(2)}}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s \mathbf{d} + \widehat{\mathbf{U}}_s \mathbf{d}_s]_2 \; [\mathbf{d}_s]_2\}_{s \in S} \right),$$

where $k \leftarrow \mathbb{Z}_p$ and $\mathbf{a}_2^{\|} \leftarrow \mathrm{span}(\mathbf{A}_2)$.

Game$_{i^\star,1,7}$: In this game, we change the $i^\star$-th secret key to be SF key. Namely, $i^\star$-th secret key is sampled as follows:

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0 \mathbf{d} + \mathbf{k}^{(2)}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s \mathbf{d} + \widehat{\mathbf{U}}_s \mathbf{d}_s]_2 \; [\mathbf{d}_s]_2\}_{s \in S} \right).$$

Note that $\mathbf{Game}_{i^\star,1,7}$ is equivalent to $\mathbf{Game}_{i^\star,2}$. Therefore, to complete the proof, it suffices to show the following lemmas. In the following, we denote the advantage of $\mathcal{A}$ in $\mathbf{Game}_{\mathrm{xx}}$ by A_{xx}.

Lemma C.1.16. *For $i^\star \in [q]$, we have $\mathsf{A}_{i^\star,1,0} = \mathsf{A}_{i^\star,1,1}$ unconditionally.*

Proof. Here, we replace $\mathbf{c} \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{A}_2)$ and $\mathbf{c}_j \leftarrow \mathrm{span}(\mathbf{A}_1, \mathbf{A}_2)$ with $\mathbf{c} + c\mathbf{a}_2$ and $\mathbf{c}_j + \mathbf{a}_{2,j}$ such that $\mathbf{c}, \mathbf{c}_j \leftarrow \mathrm{span}(\mathbf{A}_1)$, $\mathbf{a}_2, \mathbf{a}_{2,j} \leftarrow \mathrm{span}(\mathbf{A}_2)$. This clearly does not change the distribution and the lemma follows. □

Lemma C.1.17. *For $i^\star \in [q]$, we have $\mathsf{A}_{i^\star,1,1} = \mathsf{A}_{i^\star,1,2}$ unconditionally.*

Proof. We claim that if we replace $\mathbf{V}_s^{(2)}$ and $\mathbf{U}_0$ with $\mathbf{V}_s^{(2)} + v_s\mathbf{a}_2^{\|}\mathbf{b}^{\|\top}$ and $\mathbf{U}_0 + u_0\mathbf{a}_2^{\|}\mathbf{b}^{\|\top}$ in $\mathbf{Game}_{i^\star,1,1}$, the resulting distribution is the same as that of $\mathbf{Game}_{i^\star,1,2}$. Since this substitution does not change the view of the adversary, this implies the lemma. First, we observe that $\mathbf{A}_1^\top(\mathbf{U}_0 + u_0\mathbf{a}_2^{\|}\mathbf{b}^{\|\top}) = \mathbf{A}\mathbf{U}_0$ and thus the distribution of mpk is the same as that of $\mathbf{Game}_{i^\star,1,2}$. As for the keys, we have

$$\mathbf{k} + (\mathbf{U}_0 + u_0\mathbf{a}_2^{\|}\mathbf{b}^{\|\top})\mathbf{d} = \mathbf{k} + \mathbf{U}_0\mathbf{d} + u_0(\mathbf{b}^{\|\top}\mathbf{d})\mathbf{a}_2^{\|}$$

and similarly,

$$\left(\widehat{\mathbf{V}}_s + v_s\mathbf{a}_2^{\|}\mathbf{b}^{\|\top}\right)\mathbf{d} + \widehat{\mathbf{U}}_s\mathbf{d}_s = \widehat{\mathbf{V}}_s\mathbf{d} + \widehat{\mathbf{U}}_s\mathbf{d}_s + v_s(\mathbf{b}^{\|\top}\mathbf{d})\mathbf{a}_2^{\|}.$$

In the case of i-th key for $i \neq i^\star$ (namely, both for E-normal and SF keys), we have $\mathbf{b}^{\|\top}\mathbf{d} = 0$ because $\mathbf{d} \leftarrow \mathrm{span}(\mathbf{B})$. Therefore, we can see that this corresponds to the distribution of the secret key in $\mathbf{Game}_{i^\star,1,2}$.

As for the ciphertext, we have

$$\begin{aligned}\mathbf{C}_{1,j} &= \mathbf{L}_j\left(\begin{smallmatrix}(\mathbf{c}^\top + c\mathbf{a}_2^\top)(\mathbf{U}_0 + u_0\mathbf{a}_2^{\|}\mathbf{b}^{\|\top})\\ \mathbf{U}\end{smallmatrix}\right) + (\mathbf{c}_j + \mathbf{a}_{2,j})^\top\left(\widehat{\mathbf{V}}_{\rho(j)} + v_{\rho(j)}\mathbf{a}_2^{\|}\mathbf{b}^{\|\top}\right)\\ &= \mathbf{L}_j\left(\begin{smallmatrix}(\mathbf{c}^\top + c\mathbf{a}_2^\top)\mathbf{U}_0\\ \mathbf{U}\end{smallmatrix}\right) + \mathbf{a}_2^\top\mathbf{a}_2^{\|}\cdot\mathbf{L}_j\left(\begin{smallmatrix}cu_0\mathbf{b}^{\|\top}\\ \mathbf{0}\end{smallmatrix}\right) + (\mathbf{c}_j + \mathbf{a}_{2,j})^\top\widehat{\mathbf{V}}_{\rho(j)} + v_{\rho(j)}\mathbf{a}_{2,j}^\top\mathbf{a}_2^{\|}\mathbf{b}^{\|\top},\end{aligned}$$

where we use $\mathbf{c}^\top\mathbf{a}_2^{\|} = 0$ and $\mathbf{c}_j^\top\mathbf{a}_2^{\|} = 0$ in the second equation, which follow from $\mathbf{c}, \mathbf{c}_j \leftarrow \mathrm{span}(\mathbf{A}_1)$. Again, the distribution of the ciphertext corresponds to that of $\mathbf{Game}_{i^\star,1,2}$. This completes the proof of the lemma. □

Lemma C.1.18. *For $i^\star \in [q]$, we have $\mathsf{A}_{i^\star,1,2} = \mathsf{A}_{i^\star,1,3}$ unconditionally.*

Proof. We first observe that even if we replace $\mathbf{U}$ with $\mathbf{U} + \mathbf{a}_2^\top \mathbf{a}_2^{\|} \cdot \mathbf{u}\mathbf{b}^{\|\top}$ in $\mathbf{Game}_{i^\star,1,2}$, the distribution is unchanged. By rearranging the terms and substituting c_j with $(\mathbf{a}_2^\top \mathbf{a}_2^{\|})^{-1}\mathbf{a}_{2,j}^\top \mathbf{a}_2^{\|}$ in $\mathbf{Game}_{i^\star,1,3}$, we can see that $\mathbf{C}_{1,j}$ in both games are actually the same. Furthermore, since $\mathbf{a}_{2,j}^\top \mathbf{a}_2^{\|}$ is distributed uniformly at random over $\mathbb{Z}_p$ for random $\mathbf{a}_{2,j}$ sampled from $\mathrm{span}(\mathbf{A}_2)$ and $\mathbf{a}_2^\top \mathbf{a}_2^{\|} \neq 0$, the distribution of $\mathbf{a}_{2,j}$ is unchanged even if we first sample $c_j \leftarrow \mathbb{Z}_p$ and then sample it conditioned on $\mathbf{a}_{2,j}^\top \mathbf{a}_2^{\|} = (\mathbf{a}_2^\top \mathbf{a}_2^{\|})c_j$. Therefore, these games are actually equivalent and the lemma follows. $\square$

Lemma C.1.19. *For* $i^\star \in [q]$*, we have* $|\mathsf{A}_{i^\star,1,3} - \mathsf{A}_{i^\star,1,4}| = \mathrm{negl}(\lambda)$ *under the DLIN assumption.*

Proof. We assume an adversary $\mathcal{A}$ who distinguishes the games and construct another adversary $\mathcal{B}$ who distinguishes the two distributions in Lemma C.1.14. $\mathcal{B}$ first samples mpk and msk, $\mathbf{k}^{(2)}$, as well as $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{A}_2^{\|}$, $\mathbf{A}_3^{\|}$, $\mathbf{b}^{\|}$ such that $\mathbf{B}\mathbf{b}^{\|} = \mathbf{0}$. $\mathcal{B}$ also samples $\mathbf{U}_s^{(2)}$ and $\mathbf{V}_s^{(2)}$ for $s \in [n]$, where n is the upper bound on the running time of $\mathcal{A}$. $\mathcal{B}$ then gives mpk to $\mathcal{A}$, who then specifies the key queries and the attribute S for the challenge ciphertext. Let the $i^\star$-th key query made by $\mathcal{A}$ be $(\mathbf{L}, \rho)$. Then, $\mathcal{B}$ declares S and $(\mathbf{L}, \rho)$ as its target and then is given the problem instance $(\mathsf{sk}, \mathsf{ct})$. $\mathcal{B}$ generates the secret keys for $\mathcal{A}$ as specified by the game except for the $i^\star$-th key.

We then describe how $\mathcal{B}$ embeds the problem instance into the $i^\star$-th key using $\mathsf{sk} = ([\Phi]_2, \{[v_s]_2\}_{s\in S})$ from the problem instance, where $\Phi = u_0$ or $\Phi \leftarrow \mathbb{Z}_p$. It samples $\mathbf{d}, \mathbf{d}_s \leftarrow \mathbb{Z}_p^{k+1}$ for $s \in S$ and computes the $i^\star$-th key as

$$\mathsf{sk}_S := \left([\mathbf{k} + \mathbf{U}_0\mathbf{d} + \Phi(\mathbf{b}^{\|\top}\mathbf{d})\mathbf{a}_2^{\|}]_2, [\mathbf{d}]_2, \{[\widehat{\mathbf{V}}_s\mathbf{d} + \widehat{\mathbf{U}}_s\mathbf{d}_s + v_s(\mathbf{b}^{\|\top}\mathbf{d})\mathbf{a}_2^{\|}]_2\ [\mathbf{d}_s]_2\}_{s\in S}\right).$$

It is clear that the above terms are efficiently computable from sk. Furthermore, we can see that $\mathcal{B}$ simulates the $i^\star$-th key in $\mathbf{Game}_{i^\star,1,3}$ if the problem instance comes from the left distribution and $\mathbf{Game}_{i^\star,1,4}$ otherwise.

We then describe how $\mathcal{B}$ simulates the challenge ciphertext using the problem instance ct. $\mathcal{B}$ samples $\mathbf{c}, \mathbf{c}_j \leftarrow \mathrm{span}(\mathbf{A}_1)$ for $j \in [\ell]$, $\mathbf{a}_2 \leftarrow \mathrm{span}(\mathbf{A}_2)$ and $\mathbf{a}_2^{\|} \leftarrow \mathrm{span}(\mathbf{A}_2^{\|})$. $\mathcal{B}$ then computes $C_0 = [\mathbf{c}^\top + c \cdot \mathbf{a}_2^\top]_1$ and $C = e([\mathbf{c}^\top + c \cdot \mathbf{a}_2^\top]_1, [\mathbf{k}]_2) \cdot \mu_\beta$ from $[c]_1$. We then observe that $[\mathbf{a}_{2,j}]_1$ can be sampled by first sampling $\mathbf{a}_{2,j}'$ such that $\mathbf{a}_{2,j}'^\top \mathbf{a}_2^{\|} = \mathbf{a}_2^\top \mathbf{a}_2^{\|}$ and then compute $[\mathbf{a}_{2,j}]_1 := [(\mathbf{a}_{2,j}')c_j]_1$ from $[c_j]_1$. We therefore can simulate $C_{2,j} = [\mathbf{c}_j + \mathbf{a}_{2,j}]_1$ using $[\mathbf{a}_{2,j}]_1$. We finally observe that $C_{1,j} = [\mathbf{C}_{1,j}]_1$ can be efficiently computable from $[c]_1$ and $[\mathbf{L}_j \left(\begin{smallmatrix} cu_0 \\ \mathbf{u} \end{smallmatrix}\right) + c_j v_{\rho(j)}]_1$, and $[\mathbf{a}_{2,j}]_1$.

This completes the proof of the lemma. $\square$

Lemma C.1.20. *For $i^\star \in [q]$, we have $\mathsf{A}_{i^\star,1,4} = \mathsf{A}_{i^\star,1,5}$ unconditionally.*

Proof. To prove this, we undo the changes we added from $\mathbf{Game}_{i^\star,1,0}$ to $\mathbf{Game}_{i^\star,1,3}$ in the reverse order, except that $\mathbf{k}$ in the $i^\star$-th secret key is replaced with $\mathbf{k} + k\mathbf{a}_2^{\|}$. Note that all the proofs proving the (statistical) indistinguishability of the neighbouring games carry over even if the distinguisher is given $\mathbf{a}_2^{\|}$. $\square$

Lemma C.1.21. *For $i^\star \in [q]$, we have $\mathsf{A}_{i^\star,1,5} = \mathsf{A}_{i^\star,1,6}$ unconditionally.*

Proof. First observe that $\mathbf{a}_2^{\|}$ is used only in the $i^\star$-th key query and not used anywhere else. Furthermore, the distribution of $k\mathbf{a}_2^{\|}$ and $k\mathbf{a}_2^{\|} + \mathbf{k}^{(2)}$ for $\mathbf{a}_2^{\|} \leftarrow \mathrm{span}(\mathbf{A}^{(2)})$ and $k \leftarrow \mathbb{Z}_p$ are the same. By these observations, it follows that these games are actually equivalent. $\square$

Lemma C.1.22. *For $i^\star \in [q]$, we have $|\mathsf{A}_{i^\star,1,6} - \mathsf{A}_{i^\star,1,7}| = \mathrm{negl}(\lambda)$ under the DLIN assumption.*

Proof. To prove this, we undo the changes we added from $\mathbf{Game}_{i^\star,1,0}$ to $\mathbf{Game}_{i^\star,1,5}$ in the reverse order, except that $\mathbf{k}$ in the $i^\star$-th secret key is now replaced with $\mathbf{k} + \mathbf{k}^{(2)}$. $\square$

$\square$

REFERENCES

1. **Abdalla, M.**, **F. Bourse**, **A. D. Caro**, and **D. Pointcheval**, (2015). Simple Functional Encryption Schemes for Inner Products. *In Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*. 2015. URL `https://doi.org/10.1007/978-3-662-46447-2_33`.

2. **Agrawal, S.**, (2017). Stronger Security for Reusable Garbled Circuits, General Definitions and Attacks. *In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. 2017. URL `https://doi.org/10.1007/978-3-319-63688-7_1`.

3. **Agrawal, S.**, (2019). Indistinguishability Obfuscation Without Multilinear Maps: New Methods for Bootstrapping and Instantiation. *In Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*. 2019. URL `https://doi.org/10.1007/978-3-030-17653-2_7`.

4. **Agrawal, S.** and **M. Chase**, (2016). A Study of Pair Encodings: Predicate Encryption in Prime Order Groups. *In Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*. 2016. URL `https://doi.org/10.1007/978-3-662-49099-0_10`.

5. **Agrawal, S.** and **M. Chase**, (2017). FAME: Fast Attribute-based Message Encryption. *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017. URL `https://doi.org/10.1145/3133956.3134014`.

6. **Agrawal, S.**, **D. M. Freeman**, and **V. Vaikuntanathan**, (2011). Functional Encryption for Inner Product Predicates from Learning with Errors. *In Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. 2011. URL `https://doi.org/10.1007/978-3-642-25385-0_2`.

7. **Agrawal, S.**, **S. Gorbunov**, **V. Vaikuntanathan**, and **H. Wee**, (2013). Functional Encryption: New Perspectives and Lower Bounds. *In Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. 2013. URL `https://doi.org/10.1007/978-3-642-40084-1_28`.

8. **Agrawal, S.**, **B. Libert**, and **D. Stehlé**, (2016). Fully Secure Functional Encryption for Inner Products, from Standard Assumptions. *In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*. 2016. URL `https://doi.org/10.1007/978-3-662-53015-3_12`.

9. **Agrawal, S.** and **M. Maitra**, (2018). FE and iO for Turing Machines from Minimal Assumptions. *In Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*. 2018. URL `https://doi.org/10.1007/978-3-030-03810-6_18`.

10. **Agrawal, S.**, **M. Maitra**, and **S. Yamada**, (2019*a*). Attribute Based Encryption (and more) for Nondeterministic Finite Automata from LWE. *In Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*. 2019*a*. URL `https://doi.org/10.1007/978-3-030-26951-7_26`.

11. **Agrawal, S.**, **M. Maitra**, and **S. Yamada**, (2019*b*). Attribute Based Encryption for Deterministic Finite Automata from DLIN. *In Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*. 2019*b*. URL `https://doi.org/10.1007/978-3-030-36033-7_4`.

12. **Agrawal, S.** and **A. Rosen**, (2017). Functional Encryption for Bounded Collusions, Revisited. *In Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*. 2017. URL `https://doi.org/10.1007/978-3-319-70500-2_7`.

13. **Agrawal, S.** and **I. P. Singh**, (2017). Reusable Garbled Deterministic Finite Automata from Learning With Errors. *In 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*. 2017. URL `https://doi.org/10.4230/LIPIcs.ICALP.2017.36`.

14. **Ananth, P.**, **Z. Brakerski**, **G. Segev**, and **V. Vaikuntanathan**, (2015*a*). From Selective to Adaptive Security in Functional Encryption. *In Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. 2015*a*. URL `https://doi.org/10.1007/978-3-662-48000-7_32`.

15. **Ananth, P.**, **Y. Chen**, **K. Chung**, **H. Lin**, and **W. Lin**, (2016). Delegating RAM Computations with Adaptive Soundness and Privacy. *In Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. 2016. URL `https://doi.org/10.1007/978-3-662-53644-5_1`.

16. **Ananth, P.**, **X. Fan**, and **E. Shi**, (2019). Towards Attribute-Based Encryption for RAMs from LWE: Sub-linear Decryption, and More. *In Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*. 2019. URL `https://doi.org/10.1007/978-3-030-34578-5_5`.

17. **Ananth, P.** and **A. Jain**, (2015). Indistinguishability Obfuscation from Compact Functional Encryption. *In Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*. 2015. URL `https://doi.org/10.1007/978-3-662-47989-6_15`.

18. **Ananth, P.**, **A. Jain**, **D. Khurana**, and **A. Sahai**, (2018). Indistinguishability Obfuscation Without Multilinear Maps: iO from LWE, Bilinear Maps, and Weak Pseudorandomness. *IACR Cryptology ePrint Archive*, 2018. URL `https://eprint.iacr.org/2018/615`.

19. **Ananth, P.**, **A. Jain**, and **A. Sahai**, (2015*b*). Achieving Compactness Generically: Indistinguishability Obfuscation from Non-Compact Functional Encryption. *IACR Cryptology ePrint Archive*, 2015*b*. URL `http://eprint.iacr.org/2015/730`.

20. **Ananth, P.**, **A. Jain**, and **A. Sahai**, (2017). Indistinguishability Obfuscation for Turing Machines: Constant Overhead and Amortization. *In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*. 2017. URL `https://doi.org/10.1007/978-3-319-63715-0_9`.

21. **Ananth, P.** and **A. Sahai**, (2017). Projective Arithmetic Functional Encryption and Indistinguishability Obfuscation from Degree-5 Multilinear Maps. *In Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. 2017. URL `https://doi.org/10.1007/978-3-319-56620-7_6`.

22. **Ananth, P. V.** and **A. Sahai**, (2016). Functional Encryption for Turing Machines. *In Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*. 2016. URL `https://doi.org/10.1007/978-3-662-49096-9_6`.

23. **Apon, D.**, **N. Döttling**, **S. Garg**, and **P. Mukherjee**, (2017). Cryptanalysis of Indistinguishability Obfuscations of Circuits over GGH13. *In 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*. 2017. URL `https://doi.org/10.4230/LIPIcs.ICALP.2017.38`.

24. **Applebaum, B.**, **Y. Ishai**, and **E. Kushilevitz**, (2014). How to Garble Arithmetic Circuits. *SIAM J. Comput.*, **43**(2), 905–929. 2014. URL `https://doi.org/10.1137/120875193`.

25. **Arora, S.** and **B. Barak**, (2009). *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0-521-42426-4. URL `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

26. **Attrapadung, N.**, (2014). Dual System Encryption via Doubly Selective Security: Framework, Fully Secure Functional Encryption for Regular Languages, and More. *In Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*. 2014. URL `https://doi.org/10.1007/978-3-642-55220-5_31`.

27. **Attrapadung, N.**, (2016). Dual System Encryption Framework in Prime-Order Groups via Computational Pair Encodings. *In Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*. 2016. URL `https://doi.org/10.1007/978-3-662-53890-6_20`.

28. **Attrapadung, N.**, **G. Hanaoka**, and **S. Yamada**, (2015). Conversions Among Several Classes of Predicate Encryption and Applications to ABE with Various Compactness Tradeoffs. *In Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*. 2015. URL `https://doi.org/10.1007/978-3-662-48797-6_24`.

29. **Badrinarayanan, S.**, **D. Gupta**, **A. Jain**, and **A. Sahai**, (2015). Multi-input Functional Encryption for Unbounded Arity Functions. *In Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*. 2015. URL `https://doi.org/10.1007/978-3-662-48797-6_2`.

30. **Baltico, C. E. Z.**, **D. Catalano**, **D. Fiore**, and **R. Gay**, (2017). Practical Functional Encryption for Quadratic Functions with Applications to Predicate Encryption. *In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. 2017. URL `https://doi.org/10.1007/978-3-319-63688-7_3`.

31. **Barak, B.**, **O. Goldreich**, **R. Impagliazzo**, **S. Rudich**, **A. Sahai**, **S. P. Vadhan**, and **K. Yang**, (2001). On the (Im)possibility of Obfuscating Programs. *In Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. 2001. URL `https://doi.org/10.1007/3-540-44647-8_1`.

32. **Barrington, D. A. M.** (1989). Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *J. Comput. Syst. Sci.*, **38**(1), 150–164. 1989. URL `https://doi.org/10.1016/0022-0000(89)90037-8`.

33. **Bethencourt, J.**, **A. Sahai**, and **B. Waters**, (2007). Ciphertext-Policy Attribute-Based Encryption. *In 2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*. 2007. URL `https://doi.org/10.1109/SP.2007.11`.

34. **Bitansky, N.**, **S. Garg**, **H. Lin**, **R. Pass**, and **S. Telang**, (2015*a*). Succinct Randomized Encodings and their Applications. *In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 2015*a*. URL `https://doi.org/10.1145/2746539.2746574`.

35. **Bitansky, N.**, **R. Nishimaki**, **A. Passelègue**, and **D. Wichs**, (2016). From Cryptomania to Obfustopia Through Secret-Key Functional Encryption. *In Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. 2016. URL `https://doi.org/10.1007/978-3-662-53644-5_15`.

36. **Bitansky, N.**, **O. Paneth**, and **A. Rosen**, (2015*b*). On the Cryptographic Hardness of Finding a Nash Equilibrium. *In IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. 2015*b*. URL `https://doi.org/10.1109/FOCS.2015.94`.

37. **Bitansky, N.** and **V. Vaikuntanathan**, (2015). Indistinguishability Obfuscation from Functional Encryption. *In IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. 2015. URL `https://doi.org/10.1109/FOCS.2015.20`.

38. **Boneh, D.**, **C. Gentry**, **S. Gorbunov**, **S. Halevi**, **V. Nikolaenko**, **G. Segev**, **V. Vaikuntanathan**, and **D. Vinayagamurthy**, (2014). Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits. *In Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications*

of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. 2014. URL `https://doi.org/10.1007/978-3-642-55220-5_30`.

39. **Boneh, D.** and **M. Hamburg**, (2008). Generalized Identity Based and Broadcast Encryption Schemes. *In Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings.* 2008. URL `https://doi.org/10.1007/978-3-540-89255-7_28`.

40. **Boneh, D.**, **A. Sahai**, and **B. Waters**, (2011). Functional Encryption: Definitions and Challenges. *In Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings.* 2011. URL `https://doi.org/10.1007/978-3-642-19571-6_16`.

41. **Boneh, D.** and **B. Waters**, (2007). Conjunctive, Subset, and Range Queries on Encrypted Data. *In Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings.* 2007. URL `https://doi.org/10.1007/978-3-540-70936-7_29`.

42. **Boneh, D.** and **B. Waters**, (2013). Constrained Pseudorandom Functions and Their Applications. *In Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II.* 2013. URL `https://doi.org/10.1007/978-3-642-42045-0_15`.

43. **Boyen, X.** and **Q. Li**, (2015). Attribute-Based Encryption for Finite Automata from LWE. *In Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings.* 2015. URL `https://doi.org/10.1007/978-3-319-26059-4_14`.

44. **Boyle, E.**, **S. Goldwasser**, and **I. Ivan**, (2014). Functional Signatures and Pseudorandom Functions. *In Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings.* 2014. URL `https://doi.org/10.1007/978-3-642-54631-0_29`.

45. **Brakerski, Z.**, **C. Gentry**, and **V. Vaikuntanathan**, (2012). (Leveled) Fully Homomorphic Encryption without Bootstrapping. *In Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012.* 2012. URL `https://doi.org/10.1145/2090236.2090262`.

46. **Brakerski, Z.**, **I. Komargodski**, and **G. Segev**, (2016). Multi-input Functional Encryption in the Private-Key Setting: Stronger Security from Weaker Assumptions. *In Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II.* 2016. URL `https://doi.org/10.1007/978-3-662-49896-5_30`.

47. **Brakerski, Z.** and **V. Vaikuntanathan**, (2014). Lattice-based FHE as secure as PKE. *In Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014.* 2014. URL `https://doi.org/10.1145/2554797.2554799`.

48. **Brakerski, Z.** and **V. Vaikuntanathan**, (2016). Circuit-ABE from LWE: Unbounded Attributes and Semi-adaptive Security. *In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*. 2016. URL `https://doi.org/10.1007/978-3-662-53015-3_13`.

49. **Cambridge-Analytica**, (2018). Facebook-Cambridge Analytica Data Scandal. URL `https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal`.

50. **Canetti, R.**, **Y. Chen**, **J. Holmgren**, and **M. Raykova**, (2016). Adaptive Succinct Garbled RAM or: How to Delegate Your Database. *In Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. 2016. URL `https://doi.org/10.1007/978-3-662-53644-5_3`.

51. **Canetti, R.** and **J. Holmgren**, (2016). Fully Succinct Garbled RAM. *In Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*. 2016. URL `https://doi.org/10.1145/2840728.2840765`.

52. **Canetti, R.**, **J. Holmgren**, **A. Jain**, and **V. Vaikuntanathan**, (2014). Indistinguishability Obfuscation of Iterated Circuits and RAM Programs. *IACR Cryptology ePrint Archive*, 2014. URL `http://eprint.iacr.org/2014/769`.

53. **Canetti, R.**, **H. Lin**, **S. Tessaro**, and **V. Vaikuntanathan**, (2015). Obfuscation of Probabilistic Circuits and Applications. *In Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*. 2015. URL `https://doi.org/10.1007/978-3-662-46497-7_19`.

54. **Carmer, B.**, **A. J. Malozemoff**, and **M. Raykova**, (2017). 5Gen-C: Multi-input Functional Encryption and Program Obfuscation for Arithmetic Circuits. *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017. URL `https://doi.org/10.1145/3133956.3133983`.

55. **Caro, A. D.**, **V. Iovino**, **A. Jain**, **A. O'Neill**, **O. Paneth**, and **G. Persiano**, (2013). On the Achievability of Simulation-Based Security for Functional Encryption. *In Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. 2013. URL `https://doi.org/10.1007/978-3-642-40084-1_29`.

56. **Chen, J.**, **R. Gay**, and **H. Wee**, (2015*a*). Improved Dual System ABE in Prime-Order Groups via Predicate Encodings. *In Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. 2015*a*. URL `https://doi.org/10.1007/978-3-662-46803-6_20`.

57. **Chen, J.**, **J. Gong**, **L. Kowalczyk**, and **H. Wee**, (2018). Unbounded ABE via Bilinear Entropy Expansion, Revisited. *In Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*. 2018. URL `https://doi.org/10.1007/978-3-319-78381-9_19`.

58. **Chen, J.** and **H. Wee**, (2013). Fully, (Almost) Tightly Secure IBE and Dual System Groups. *In Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. 2013. URL `https://doi.org/10.1007/978-3-642-40084-1_25`.

59. **Chen, J.** and **H. Wee**, (2014). Semi-adaptive Attribute-Based Encryption and Improved Delegation for Boolean Formula. *In Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*. 2014. URL `https://doi.org/10.1007/978-3-319-10879-7_16`.

60. **Chen, Y.**, **S. S. M. Chow**, **K. Chung**, **R. W. F. Lai**, **W. Lin**, and **H. Zhou**, (2015*b*). Computation-Trace Indistinguishability Obfuscation and its Applications. *IACR Cryptology ePrint Archive*, 2015. URL `http://eprint.iacr.org/2015/406`.

61. **Cheon, J. H.**, **P. Fouque**, **C. Lee**, **B. Minaud**, and **H. Ryu**, (2016*a*). Cryptanalysis of the New CLT Multilinear Map over the Integers. *In Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*. 2016*a*. URL `https://doi.org/10.1007/978-3-662-49890-3_20`.

62. **Cheon, J. H.**, **K. Han**, **C. Lee**, **H. Ryu**, and **D. Stehlé**, (2015). Cryptanalysis of the Multilinear Map over the Integers. *In Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. 2015. URL `https://doi.org/10.1007/978-3-662-46800-5_1`.

63. **Cheon, J. H.**, **J. Jeong**, and **C. Lee**, (2016*b*). An Algorithm for NTRU Problems and Cryptanalysis of the GGH Multilinear Map without an encoding of zero. *IACR Cryptology ePrint Archive*, 2016. URL `http://eprint.iacr.org/2016/139`.

64. **Cohen, A.**, **J. Holmgren**, **R. Nishimaki**, **V. Vaikuntanathan**, and **D. Wichs**, (2016). Watermarking Cryptographic Capabilities. *In Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. 2016. URL `https://doi.org/10.1145/2897518.2897651`.

65. **Cook, S. A.** and **H. J. Hoover**, (1985). A Depth-Universal Circuit. *SIAM J. Comput.*, **14**(4), 833–839. 1985. URL `https://doi.org/10.1137/0214058`.

66. **Coron, J.**, **C. Gentry**, **S. Halevi**, **T. Lepoint**, **H. K. Maji**, **E. Miles**, **M. Raykova**, **A. Sahai**, and **M. Tibouchi**, (2015). Zeroizing Without Low-Level Zeroes: New MMAP Attacks and their Limitations. *In Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*. 2015. URL `https://doi.org/10.1007/978-3-662-47989-6_12`.

67. **Coron, J.**, **M. S. Lee**, **T. Lepoint**, and **M. Tibouchi**, (2017). Zeroizing Attacks on Indistinguishability Obfuscation over CLT13. *In Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*. 2017. URL `https://doi.org/10.1007/978-3-662-54365-8_3`.

68. **Diffie, W.** and **M. E. Hellman**, (1976). New Directions in Cryptography. *IEEE Trans. Information Theory*, **22**(6), 644–654. 1976. URL `https://doi.org/10.1109/TIT.1976.1055638`.

69. **Doe, D.** (2019). Two more leaks expose Indian citizens' personal and medical information. URL https://www.databreaches.net/two-more-leaks-expose-indian-citizens-personal-and-medical-information/.

70. **Escala, A.**, **G. Herold**, **E. Kiltz**, **C. Ràfols**, and **J. Villar** (2017). An algebraic framework for diffie–hellman assumptions. *Journal of cryptology*, **30**(1), 242–288.

71. **Garg, S.**, **C. Gentry**, **S. Halevi**, **M. Raykova**, **A. Sahai**, and **B. Waters**, (2013*a*). Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. *In 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. 2013*a*. URL https://doi.org/10.1109/FOCS.2013.13.

72. **Garg, S.**, **C. Gentry**, **S. Halevi**, **A. Sahai**, and **B. Waters**, (2013*b*). Attribute-Based Encryption for Circuits from Multilinear Maps. *In Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. 2013*b*. URL https://doi.org/10.1007/978-3-642-40084-1_27.

73. **Garg, S.**, **O. Pandey**, and **A. Srinivasan**, (2016). Revisiting the Cryptographic Hardness of Finding a Nash Equilibrium. *In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*. 2016. URL https://doi.org/10.1007/978-3-662-53008-5_20.

74. **Garg, S.**, **O. Pandey**, **A. Srinivasan**, and **M. Zhandry**, (2017). Breaking the Sub-Exponential Barrier in Obfustopia. *In Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*. 2017. URL https://doi.org/10.1007/978-3-319-56617-7_6.

75. **Garg, S.** and **A. Srinivasan**, (2016). Single-Key to Multi-Key Functional Encryption with Polynomial Loss. *In Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. 2016. URL https://doi.org/10.1007/978-3-662-53644-5_16.

76. **Gentry, C.**, **S. Halevi**, **M. Raykova**, and **D. Wichs**, (2014). Outsourcing Private RAM Computation. *In 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. 2014. URL https://doi.org/10.1109/FOCS.2014.50.

77. **Gentry, C.**, **A. Sahai**, and **B. Waters**, (2013). Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. *In Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. 2013. URL https://doi.org/10.1007/978-3-642-40041-4_5.

78. **Goldreich, O.**, **S. Goldwasser**, and **S. Micali**, (1986). How to Construct Random Functions. *J. ACM*, **33**(4), 792–807. 1986. URL https://doi.org/10.1145/6490.6503.

79. **Goldwasser, S.**, **S. D. Gordon**, **V. Goyal**, **A. Jain**, **J. Katz**, **F. Liu**, **A. Sahai**, **E. Shi**, and **H. Zhou**, (2014). Multi-input Functional Encryption. *In Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications*

of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. 2014. URL https://doi.org/10.1007/978-3-642-55220-5_32.

80. **Goldwasser, S.**, **Y. T. Kalai**, **R. A. Popa**, **V. Vaikuntanathan**, and **N. Zeldovich**, (2013*a*). How to Run Turing Machines on Encrypted Data. *In Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. 2013*a*. URL https://doi.org/10.1007/978-3-642-40084-1_30.

81. **Goldwasser, S.**, **Y. T. Kalai**, **R. A. Popa**, **V. Vaikuntanathan**, and **N. Zeldovich**, (2013*b*). Reusable Garbled Circuits and Succinct Functional Encryption. *In Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. 2013*b*. URL https://doi.org/10.1145/2488608.2488678.

82. **Gong, J.**, **B. Waters**, and **H. Wee**, (2019). ABE for DFA from k-Lin. *In Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*. 2019. URL https://doi.org/10.1007/978-3-030-26951-7_25.

83. **Gorbunov, S.**, **V. Vaikuntanathan**, and **H. Wee**, (2012). Functional Encryption with Bounded Collusions via Multi-party Computation. *In Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. 2012. URL https://doi.org/10.1007/978-3-642-32009-5_11.

84. **Gorbunov, S.**, **V. Vaikuntanathan**, and **H. Wee**, (2013). Attribute-based Encryption for Circuits. *In Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. 2013. URL https://doi.org/10.1145/2488608.2488677.

85. **Gorbunov, S.**, **V. Vaikuntanathan**, and **H. Wee**, (2015). Predicate Encryption for Circuits from LWE. *In Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. 2015. URL https://doi.org/10.1007/978-3-662-48000-7_25.

86. **Gorbunov, S.** and **D. Vinayagamurthy**, (2015). Riding on Asymmetry: Efficient ABE for Branching Programs. *In Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*. 2015. URL https://doi.org/10.1007/978-3-662-48797-6_23.

87. **Goyal, R.**, **V. Koppula**, and **B. Waters**, (2016). Semi-adaptive Security and Bundling Functionalities Made Generic and Easy. *In Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. 2016. URL https://doi.org/10.1007/978-3-662-53644-5_14.

88. **Goyal, V.**, **O. Pandey**, **A. Sahai**, and **B. Waters**, (2006). Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. *In Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*. 2006. URL https://doi.org/10.1145/1180405.1180418.

89. **Hamlin, A.**, **J. Holmgren**, **M. Weiss**, and **D. Wichs**, (2019). On the Plausibility of Fully Homomorphic Encryption for RAMs. *In Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*. 2019. URL https://doi.org/10.1007/978-3-030-26948-7_21.

90. **Hu, Y.** and **H. Jia**, (2015). Cryptanalysis of GGH Map. *IACR Cryptology ePrint Archive*, 2015. URL http://eprint.iacr.org/2015/301.

91. **Impagliazzo, R.** (2011). Notes on Turing Machines. http://cseweb.ucsd.edu/classes/sp11/cse201A-a/ln412.pdf.

92. **Jafargholi, Z.**, **A. Scafuro**, and **D. Wichs**, (2017). Adaptively Indistinguishable Garbled Circuits. *In Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*. 2017. URL https://doi.org/10.1007/978-3-319-70503-3_2.

93. **Katz, J.**, **A. Sahai**, and **B. Waters**, (2008). Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. *In Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*. 2008. URL https://doi.org/10.1007/978-3-540-78967-3_9.

94. **Kiayias, A.**, **S. Papadopoulos**, **N. Triandopoulos**, and **T. Zacharias**, (2013). Delegatable Pseudorandom Functions and Applications. *In 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 2013. URL https://doi.org/10.1145/2508859.2516668.

95. **Kitagawa, F.**, **R. Nishimaki**, and **K. Tanaka**, (2017). Indistinguishability Obfuscation for All Circuits from Secret-Key Functional Encryption. *IACR Cryptology ePrint Archive*, 2017. URL http://eprint.iacr.org/2017/361.

96. **Kitagawa, F.**, **R. Nishimaki**, and **K. Tanaka**, (2018*a*). Obfustopia Built on Secret-Key Functional Encryption. *In Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. 2018*a*. URL https://doi.org/10.1007/978-3-319-78375-8_20.

97. **Kitagawa, F.**, **R. Nishimaki**, and **K. Tanaka**, (2018*b*). Simple and Generic Constructions of Succinct Functional Encryption. *In Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*. 2018*b*. URL https://doi.org/10.1007/978-3-319-76581-5_7.

98. **Kitagawa, F.**, **R. Nishimaki**, **K. Tanaka**, and **T. Yamakawa**, (2019). Adaptively Secure and Succinct Functional Encryption: Improving Security and Efficiency, Simultaneously. *In Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*. 2019. URL https://doi.org/10.1007/978-3-030-26954-8_17.

99. **Komargodski, I.** and **G. Segev**, (2017). From Minicrypt to Obfustopia via Private-Key Functional Encryption. *In Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic*

Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I. 2017. URL https://doi.org/10.1007/978-3-319-56620-7_5.

100. **Koppula, V.**, **A. B. Lewko**, and **B. Waters**, (2015). Indistinguishability Obfuscation for Turing Machines with Unbounded Memory. *In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 2015. URL https://doi.org/10.1145/2746539.2746614.

101. **Kowalczyk, L.** and **A. B. Lewko**, (2015). Bilinear Entropy Expansion from the Decisional Linear Assumption. *In Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. 2015. URL https://doi.org/10.1007/978-3-662-48000-7_26.

102. **Kowalczyk, L.** and **H. Wee**, (2019). Compact Adaptively Secure ABE for NC1 from k-Lin. *IACR Cryptology ePrint Archive*, 2019. URL https://eprint.iacr.org/2019/224.

103. **Lewko, A. B.**, (2012). Tools for Simulating Features of Composite Order Bilinear Groups in the Prime Order Setting. *In Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*. 2012. URL https://doi.org/10.1007/978-3-642-29011-4_20.

104. **Lewko, A. B.**, **T. Okamoto**, **A. Sahai**, **K. Takashima**, and **B. Waters**, (2010). Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. *In Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*. 2010. URL https://doi.org/10.1007/978-3-642-13190-5_4.

105. **Lewko, A. B.** and **B. Waters**, (2010). New Techniques for Dual System Encryption and Fully Secure HIBE with Short Ciphertexts. *In Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*. 2010. URL https://doi.org/10.1007/978-3-642-11799-2_27.

106. **Lewko, A. B.** and **B. Waters**, (2011). Unbounded HIBE and Attribute-Based Encryption. *In Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*. 2011. URL https://doi.org/10.1007/978-3-642-20465-4_30.

107. **Lewko, A. B.** and **B. Waters**, (2012). New Proof Methods for Attribute-Based Encryption: Achieving Full Security through Selective Techniques. *In Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. 2012. URL https://doi.org/10.1007/978-3-642-32009-5_12.

108. **Li, B.** and **D. Micciancio**, (2016). Compactness vs Collusion Resistance in Functional Encryption. *In Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*. 2016. URL https://doi.org/10.1007/978-3-662-53644-5_17.

109. **Lin, H.**, (2016). Indistinguishability Obfuscation from Constant-Degree Graded Encoding Schemes. *In Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*. 2016. URL https://doi.org/10.1007/978-3-662-49890-3_2.

110. **Lin, H.**, (2017). Indistinguishability Obfuscation from SXDH on 5-Linear Maps and Locality-5 PRGs. *In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. 2017. URL https://doi.org/10.1007/978-3-319-63688-7_20.

111. **Lin, H.** and **C. Matt**, (2018). Pseudo Flawed-Smudging Generators and Their Application to Indistinguishability Obfuscation. *IACR Cryptology ePrint Archive*, 2018. URL https://eprint.iacr.org/2018/646.

112. **Lin, H.**, **R. Pass**, **K. Seth**, and **S. Telang**, (2016). Output-Compressing Randomized Encodings and Applications. *In Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*. 2016. URL https://doi.org/10.1007/978-3-662-49096-9_5.

113. **Lin, H.** and **S. Tessaro**, (2017). Indistinguishability Obfuscation from Trilinear Maps and Block-Wise Local PRGs. *In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. 2017. URL https://doi.org/10.1007/978-3-319-63688-7_21.

114. **Lin, H.** and **V. Vaikuntanathan**, (2016). Indistinguishability Obfuscation from DDH-Like Assumptions on Constant-Degree Graded Encodings. *In IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. 2016. URL https://doi.org/10.1109/FOCS.2016.11.

115. **Liu, Q.** and **M. Zhandry**, (2017). Decomposable Obfuscation: A Framework for Building Applications of Obfuscation from Polynomial Hardness. *In Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*. 2017. URL https://doi.org/10.1007/978-3-319-70500-2_6.

116. **Lu, S.** and **R. Ostrovsky**, (2013). How to Garble RAM Programs. *In Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. 2013. URL https://doi.org/10.1007/978-3-642-38348-9_42.

117. **Miles, E.**, **A. Sahai**, and **M. Zhandry**, (2016). Annihilation Attacks for Multilinear Maps: Cryptanalysis of Indistinguishability Obfuscation over GGH13. *In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*. 2016. URL https://doi.org/10.1007/978-3-662-53008-5_22.

118. **Okamoto, T.** and **K. Takashima**, (2010). Fully Secure Functional Encryption with General Relations from the Decisional Linear Assumption. *In Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August*

15-19, 2010. Proceedings. 2010. URL https://doi.org/10.1007/978-3-642-14623-7_11.

119. **Okamoto, T.** and **K. Takashima**, (2012). Fully Secure Unbounded Inner-Product and Attribute-Based Encryption. *In Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*. 2012. URL https://doi.org/10.1007/978-3-642-34961-4_22.

120. **O'Neill, A.** (2010). Definitional Issues in Functional Encryption. *IACR Cryptology ePrint Archive*, 2010. URL http://eprint.iacr.org/2010/556.

121. **Pippenger, N.** and **M. J. Fischer**, (1979). Relations Among Complexity Measures. *J. ACM*, **26**(2), 361–381. 1979. URL https://doi.org/10.1145/322123.322138.

122. **PTI**, (2019). Indian organisations lost 12.8 crore to data breaches. URL https://www.thehindu.com/business/indian-organisations-lost-128-crore-to-data-breaches/article28681416.ece.

123. **Rivest, R. L.**, **A. Shamir**, and **L. M. Adleman** (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, **21**(2), 120–126. 1978. URL http://doi.acm.org/10.1145/359340.359342.

124. **Rouselakis, Y.** and **B. Waters**, (2013). Practical Constructions and New Proof Methods for Large Universe Attribute-Based Encryption. *In 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 2013. URL https://doi.org/10.1145/2508859.2516672.

125. **Sahai, A.** and **B. Waters**, (2005). Fuzzy Identity-Based Encryption. *In Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. 2005. URL https://doi.org/10.1007/11426639_27.

126. **Sahai, A.** and **B. Waters**, (2014). How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. *In Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. 2014. URL https://doi.org/10.1145/2591796.2591825.

127. **Sipser, M.** (1996). Introduction to the Theory of Computation. *SIGACT News*, **27**(1), 27–29. 1996. URL https://doi.org/10.1145/230514.571645.

128. **Thompson, H.** and **S. Trilling**, (2018). Cyber Security Predictions: 2019 and Beyond. URL https://www.symantec.com/blogs/feature-stories/cyber-security-predictions-2019-and-beyond.

129. **Vishwanath, S.** (2019). Seven cyber security trends that India will witness in 2019: PwC's forecast. URL https://www.pwc.in/consulting/cyber-security/blogs/seven-cyber-security-trends-that-india-will-witness-in-2019.html.

130. **Waters, B.**, (2012). Functional Encryption for Regular Languages. *In Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. 2012. URL https://doi.org/10.1007/978-3-642-32009-5_14.

131. **Wee, H.** (2015). Dual System Encryption via Predicate Encodings. *IACR Cryptology ePrint Archive*, 2015. URL `http://eprint.iacr.org/2015/273`.

LIST OF PAPERS BASED ON THESIS

1. Shweta Agrawal and Monosij Maitra. “FE and iO for Turing Machines from Minimal Assumptions.” In *Theory of Cryptography Conference*, pp. 473-512. Springer, Cham, (2018).

2. Shweta Agrawal and Monosij Maitra and Shota Yamada. "Attribute Based Encryption (and more) for Nondeterministic Finite Automata from LWE." In *Annual International Cryptology Conference*, pp. 765-797, Springer, Cham, (2019).

3. Shweta Agrawal and Monosij Maitra and Shota Yamada. "Attribute Based Encryption for Deterministic Finite Automata from DLIN." In *Theory of Cryptography Conference*, pp. 91-117, Springer, Cham, (2019).

Resume

Personal Details:

Name: Monosij Maitra

Date of Birth: 19-Apr-1988

e-Mail Address: monosij.maitra@gmail.com, monosij@cse.iitm.ac.in

Permanent Address: 15B, P. Naskar Lane, Picnic Garden
Kolkata- 700039, INDIA.

Education:

PhD. (2015 - present): Indian Institute of technology Madras, Chennai, India.

M.E. (2012 - 2014): Indian Institute of Engineering Science and Technology
Shibpur, Howrah, India.

B.Tech. (2007 - 2011): Future Institute of Engineering and Management,
Kolkata, India.

Doctoral Committee

Chair Person: Prof. P. Sreenivasa Kumar - CSE Department

Guide: Dr. Shweta Agrawal - CSE Department

Members:

1. Dr. Meghana Nasre - CSE Department
2. Dr. Jayalal Sarma - CSE Department
3. Prof. Andrew Thangaraj - Department of Electrical Engineering