Differing-Inputs Obfuscation and Applications

Abstract

In this paper, we study of the notion of *differing-input obfuscation*, introduced by Barak et al. (CRYPTO 2001, JACM 2012). For any two circuits C_0 and C_1, a differing-input obfuscator $\mathsf{di}\mathcal{O}$ guarantees that the non-existence of an adversary that can find an input on which C_0 and C_1 differ implies that $\mathsf{di}\mathcal{O}(C_0)$ and $\mathsf{di}\mathcal{O}(C_1)$ are computationally indistinguishable. We show many applications of this notion:

- We define the notion of a differing-input obfuscator for Turing machines and give a construction for the same (without converting it to a circuit) with *input-specific running times*. More specifically, for each input, our obfuscated Turning machine takes time proportional to the running time of the Turing machine on that specific input rather than the machine's worst-case running time.
- We give a functional encryption scheme that allows for secret-keys to be associated with Turing machines, and thereby achieves *input-specific running times*. Further, we can equip our functional encryption scheme with *delegation* properties.
- We construct a multi-party non-interactive key exchange protocol with no trusted setup where all parties post only logarithmic-size messages. It is the first such scheme with such short messages. We similarly obtain a broadcast encryption system where the ciphertext overhead and secret-key size is constant (i.e. independent of the number of users), and the public key is logarithmic in the number of users.

All our constructions make inherent use of the power provided by differing-input obfuscation. It is not currently known how to construct systems with these properties from the weaker notion of indistinguishability obfuscation.

1 Introduction

General-purpose program obfuscation aims to make arbitrary computer programs "unintelligible" while preserving their functionality. The first formal study of the problem of obfuscation was undertaken by Hada [Had00] and Barak et al. [BGI+12]. Barak et al. proposed the notion of *virtual black-box* (VBB) obfuscation. This notion requires that the obfuscation does not leak anything more than what can be learnt with just a *black-box* oracle access to the function. Unfortunately, in the same work, Barak et al. showed a family of circuits that cannot be VBB obfuscated.

Weaker variants Obfuscation. In light of this impossibility result, Barak et al. left open the problem of realizing weaker notions of obfuscation such as indistinguishability obfuscation and differing-inputs obfuscation (see below for further explanation).

Indistinguishability obfuscation requires that given any two equivalent circuits C_0 and C_1 of similar size, the obfuscations of C_0 and C_1 are computationally indistinguishable. In a very recent work, Garg et al. [GGH+13b], building upon the multilinear maps framework of Garg et

al. [GGH13a], gave the first candidate construction for a general-purpose obfuscator satisfying this notion.

The stronger notion of *differing-inputs* obfuscation [BGI+12] states that the existence of an adversary that can distinguish between obfuscations of circuits C_0 and C_1 implies the existence of an adversary that can actually *extract* an input on which the two circuits differ. The starting point for our work is the conjecture that the Garg et al. construction (and variants of it [BR13, BGK+13]) indeed achieves the differing-inputs obfuscation notion. Perhaps the strongest evidence for this conjecture is provided by analysis of this construction and variants in suitable generic models [GGH+13b, BR13, BGK+13].[1]

The focus of this paper is to show (1) how to bootstrap the notion of differing inputs obfuscation to build differing inputs obfuscators for Turing Machines with per-input running time, and (2) how to leverage differing inputs obfuscation to obtain a number of interesting applications. Before we turn to our main results, we first illustrate the usefulness of differing inputs obfuscation with two simple examples.

Warmup: Extractable Witness Encryption for NP. As a warmup example, we will show how differing-inputs obfuscation can be used to construct extractable witness encryption, a primitive which already has the flavor of extraction. The notion of extractable witness encryption for NP recently introduced in [GGSW13, GKP+13a] states that given an NP language L, an extractable witness encryption scheme for L is an encryption scheme that takes as input an instance x and a message bit b, and outputs a ciphertext c. If $x \in L$ and w is a valid witness for x, then a decryptor can use w to decrypt c and recover b. Furthermore, security requires that any adversary that can decrypt ciphertext c corresponding to instance x can be used to extract a valid witness for x.

Now we present our construction of extractable witness encryption, which is analogous to the construction of witness encryption from indistinguishability obfuscation as in Garg et al. [GGH+13b]. We define the circuits $C_{x,b}$ for $b \in \{0,1\}$ taking w as input as follows. If w is a valid witness for x, then $C_{x,b}$ outputs b and $\perp$ otherwise. Differing-input obfuscation of $C_{x,b}$ will serve as an encryption of the bit b. Correctness of decryption is immediate. Recall that the security of differing-inputs obfuscation states that existence of a distinguisher between the obfuscations of the two circuits implies the existence of an adversary that can find an input on which the two circuits differ. Thus, by the security of differing-inputs obfuscation, we conclude that an adversary breaking the semantic security of the above encryption scheme can be used to extract a valid witness for x.

Example of restricted use software. We find the differing-inputs obfuscation notion stated in its contrapositive form, i.e. non-existence of an adversary that can find input on which the two circuits differ implies the non-existence of an adversary that can distinguish between the obfuscations of the two circuits, as more insightful when considering applications. We highlight how this interpretation can be useful for applications such as *restricted-use software:*

Software developers often want to release multiple tiers of a product with different price points, allowing for different levels of functionality. In principle, each customer could be provided a separate version of the software, enabling only the features he needs. Ideally, a developer could just have flags corresponding to each feature in his software. The developer could then create a customized version of the software simply by starting with the full version and then turning off the features the consumer does not want directly at the interface level — requiring minimal additional effort. However, if this is all that is done, then it would be easy for an attacker to bypass these controls and gain access to the full version or the code behind it. The

[1] In particular, this line of work [GGH+13b, BR13, BGK+13] culminated in an unconditional realization of the VBB notion in the generic multilinear model. The crucial point for our case is that this proof shows extractability of differing inputs from any distinguishing adversary, though only in a generic model.

other alternative is for a software development team to carefully remove all unused components — an elaborate task. Can we have the best of both worlds? Our solution is for a developer to release an obfuscated version of the program that takes as input a signature on the custom set of functionality flags that the consumer has paid for. Next, we argue that for this application, differing-inputs obfuscation suffices. Assuming unforgeability, no efficient malicious user can generate a signature on any set of attributes besides the ones provided to it. Given that observation, differing-inputs obfuscation immediately implies that the obfuscated program with selected features turned off in the perspective of the user is indistinguishable from the obfuscation of the program with unwanted parts removed at the start.

On the other hand, note that indistinguishability obfuscation would not suffice here: This is because the program with the unwanted parts removed implements a different functionality from the original program, and therefore, indistinguishability obfuscation alone does not guarantee security in this setting.

1.1 Our Results

We obtain the following results:

Differing-input obfuscation for Turing Machines: We define the notion of differing-input obfuscator for Turing machines and give a construction for Turing machines with bounded length inputs (without converting it to a circuit), assuming the existence of a differing-input obfuscator for circuits and SNARGs for P [BCCT13]. Additionally, assuming SNARKs for P [BCCT13], we can construct a differing-input obfuscator even for the setting where the length of the input is not bounded. (We stress that it is only for this extension that we need to assume SNARKs.) Moreover, our construction achieves *input-specific running times* (explained below). This means that evaluating the obfuscated machine on input x does not depend on the worst-case running time of the machine but just on the running time of the unobfuscated machine on input x.

Input-specific runtime. Most tasks in cryptography are well suited for circuits and not for Turing machines. Hence, most cryptographic applications require that a Turing machine be first transformed to a circuit, leading to inefficiency. This is especially true when computing on encrypted data. For example, consider the case of fully-homomorphic encryption (FHE): Consider computing a specified Turing machine on an encrypted input. If we were to first convert this Turing machine to a circuit, this would mean that on every input, the evaluation will take time proportional to the worst-case running time rather than time it takes for evaluation on that specific input. The first variants of FHE that achieve these properties were given by [GKP$^+$13b, GKP$^+$13a]. We use ideas from both of these works in our constructions and achieve input-specific running times which may be substantially better than worst-case. Oblivious RAM techniques for relaxing the need to convert a Turing machine to a circuit were also explored in [OS97, LO13].

Functional Encryption for Turing Machines: We give a selectively secure functional encryption scheme that allows for secret-keys to be associated with Turing machines and thereby achieves *input-specific running times*. Further, the scheme can be equipped with *delegation* properties. We note that for the case of single-key functional encryption [SS10], the problem of supporting Turing Machines and achieving input-specific runtimes was previously introduced and resolved by Goldwasser et al. [GKP$^+$13a].

Short multiparty key exchange and broadcast encryption: In recent work, Boneh and Zhandry [BZ13] show that indistinguishability obfuscation gives a broadcast encryption system with properties that were not previously achievable (we refer to [BZ13] for a detailed

survey of related work). While ciphertexts and secret keys in their system are constant size (i.e., independent of the number of users), the size of their public-key is *linear* in the total number of users N. The reason for the linear-size public-key is an obfuscated program used for decryption that takes as input (a representation of) the recipient set $S \subseteq [N]$ and a recipient private key SK_i. The program verifies that recipient i is part of the recipient set S, and if so, outputs a ciphertext decryption key. Since the recipient set can be linear size, the obfuscated program had to be linear size, thereby forcing the public-key to be linear size.

A natural approach to shrink the decryption program in the public-key is as follows: instead of giving the program the recipient set S as an argument, we give it a short proof that i is in S. The obfuscated decryption program will check the proof, and if valid, will decrypt the given ciphertext. A simple proof for the statement $i \in S$ can be built from collision resistant hash functions using Merkle hash trees [Mer88]. Unfortunately, indistinguishability obfuscation ($\mathsf{i}\mathcal{O}$) is insufficient for proving security of this approach using current techniques. The problem is that using $\mathsf{i}\mathcal{O}$, we can only puncture a certain PRF embedded in the obfuscated program if the resulting program is *identical* to the original program. However, because the proofs for $i \in S$ are succinct, there *exist* false proofs. That is, for any set $S' \subseteq [N]$ for which $i \notin S'$, there exists a convincing (false) proof that $i \in S'$. These false proofs prevent us from applying $\mathsf{i}\mathcal{O}$ to argue that the punctured program is indistinguishable from the original program. While false proofs exist, finding a false proof will break collision resistance of the hash function used to construct the Merkle tree. Therefore, differing-inputs obfuscation can be applied because no polynomial time algorithm can distinguish the punctured program from the original program.

To make this idea work, we have to further modify the mechanism used in the broadcast system of [BZ13]. Our final construction, presented in Section 4, is such that proving security requires two applications of $\mathsf{di}\mathcal{O}$ in three hybrid games. We also show that the same idea can be used to improve the multiparty non-interactive key exchange (NIKE) from [BZ13] so that, even when there is no trusted setup, all parties post at most a logarithmic-size message (in the number of users) to the public bulletin board.

1.2 Concurrent and Independent Work

A concurrent and independent work of [BCP14] also studies differing inputs obfuscation (which they call extractable obfuscation) and obtains a number of applications for differing inputs obfuscation. This work overlaps in part with our own, but there are differences: Most notably, on the one hand, [BCP14] demonstrate a remarkable implication showing that indistinguishability obfuscators *must* satisfy a weak form of differing inputs obfuscation for any pair of circuits that only differ on a polynomial-size set of inputs[2]. Most notably on the other side, we stress that the applications of multi-party non-interactive key exchange protocol and broadcast encryption given in this paper do not appear in their work. Furthermore, their main application of obfuscating Turing machines has restrictions on the input length and running time [3] whereas our result does not have have such restrictions.

1.3 Subsequent Work

Subsequent to the posting of our work online, many applications of differing-inputs obfuscation have been studied. Assuming differing-inputs obfuscation, Goldwasser et al. [GGG+14] construct compact indistinguishability-secure multi-input FE schemes. Bellare and Tessaro [BT13] construct polynomially many hard-core bits from any one-way function assuming differing-inputs

[2]We note, however, that none of the applications we consider here would work with weak differing inputs obfuscators.

[3]We note that [BCP14] does mention how to achieve the same goal without any restrictions (on the input length and the running time). However, only a sketch of the approach is provided.

obfuscation. Using our construction of diO for Turing machines, Pandey et al. [PPS13] construct a 4-message concurrent zero knowledge protocol.

Boyle et al. [BP13] show that extractability obfuscators for Turing machines and SNARKs cannot co-exist with respect to specific (unnatural) auxiliary input distributions, and in particular, their result does not rule out the auxiliary input distributions we use in our work. Garg et al. [GGHW13] show that if a specific type of obfuscation exists (one that is not implied by differing inputs obfuscation), then differing-inputs obfuscation for circuits cannot exist with regard to specific (unnatural) auxiliary input distributions. Although we find this follow-up work to ours intriguing, the assumption that their specific type of obfuscation exists is also not a natural one [4]. Furthermore we stress that we limit ourselves to more natural families of auxiliary information that are unaffected by these results.

2 Preliminaries

2.1 Notation

We represent the security parameter by λ. A function f is said to be negligible in a variable n if for every polynomial p, we have $f(n) < \frac{1}{p(n)}$. For an algorithm A, we use the notation $o \leftarrow A(i)$ to denote that the output of A on input i is o. We use $r \xleftarrow{\$} \mathcal{S}$ to denote that r is drawn from the space $\mathcal{S}$ uniformly at random.

We assume that the reader is familiar with the concept of Turing machines. We denote the running time of Turing machine M on input x by $\mathsf{time}(M, x)$. We say that the output of two Turing machines on an input are the same if the output tapes of the two Turing machines are identical.

For every NP-language L, we associate a corresponding relation R_L such that an instance $x \in L$ iff there exists a witness w such that $(x, w) \in R_L$. Furthermore, we say an instance x is a "valid" (or a true) instance iff $x \in L$. Correspondingly, those instances that don't belong to the language are referred to as invalid (or false) statements.

2.2 Differing-inputs Obfuscation for circuits and TMs

We recall the notion of differing-inputs obfuscation from Barak et. al. [BGI+12]. Next we present this notion for both circuits and Turing machines. Before we go ahead with the definition, we describe the notion of differing-inputs circuit family. Intuitively, we call a circuit family to be differing-inputs circuit family if there does not exist any PPT adversary who given two circuits, which are sampled from a distribution defined on this circuit family, can output a value such that both the circuits differ on this input.

Definition 1. *A circuit family $\mathcal{C}$ associated with a PPT* $\mathsf{Sampler}$ *is said to be a differing-inputs circuit family if for every PPT adversary $\mathcal{A}$ there exists a negligible function α such that:*

$$\mathsf{Prob}[C_0(x) \neq C_1(x) \ : \ (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda), \ x \leftarrow \mathcal{A}(1^\lambda, C_0, C_1, \mathsf{aux})] \leq \alpha(\lambda).$$

We now define the notion of differing-inputs obfuscation for a differing-inputs circuit family.

Definition 2. ***(Differing-inputs Obfuscators for circuits)*** *A uniform PPT machine $\mathsf{di}\mathcal{O}$ is called a Differing-inputs Obfuscator for a differing-inputs circuit family $\mathcal{C} = \{C_\lambda\}$ if the following conditions are satisfied:*

- ***Correctness**: For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}$, for all inputs x, we have that*

$$\mathsf{Prob}[C'(x) = C(x) \ : \ C' \leftarrow \mathsf{di}\mathcal{O}(\lambda, C)] \ = \ 1$$

[4] Indeed, even a heuristic construction of their specific type of obfuscation is not known.

- ***Polynomial slowdown**: There exists a universal polynomial p such that for any circuit C, we have $|C'| \leq p(|C|)$, where $C' = \mathsf{di}\mathcal{O}(\lambda, C)$.*
- ***Differing-inputs**: For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda)$, we have that*

$$|\mathsf{Prob}[D(\mathsf{di}\mathcal{O}(\lambda, C_0), \mathsf{aux}) = 1] \; - \; \mathsf{Prob}[D(\mathsf{di}\mathcal{O}(\lambda, C_1), \mathsf{aux}) = 1] \leq \alpha(\lambda)$$

The concept of differing-inputs obfuscation can be thought of as a generalisation of indistinguishable obfuscation. This is because, indistinguishable obfuscation is defined for circuits which are identical on all inputs and hence such circuits trivially satisfy the definition of differing-inputs circuit familes. We conjecture that the construction defined in Garg et al. [GGH$^+$13b] and its optimizations from [BR13, BGK$^+$13] satisfies this stronger notion of differing-inputs obfuscation. In this work, using this notion we obtain many applications.

We now consider the case when we are obfuscating Turing machines. Even before we define the differing-inputs property for Turing machines we first need to define the notion of differing-inputs Turing machines family. Before we define this, we consider the family of Turing machines $\mathcal{M}$ which is equipped with $\mathsf{Sampler}_{\mathcal{M}}$ which efficiently samples two Turing machines from $\mathcal{M}$ along with auxiliary information. For simplicity, we assume that every $M \in \mathcal{M}$ on an input x outputs the time it runs in, in addition to its output.

Definition 3. *A Turing machine family $\mathcal{M}$ associated with a sampler $\mathsf{Sampler}_{\mathcal{M}}$ is said to be a differing-inputs Turing machine family if for every PPT adversary, the following holds*

$$\mathsf{Prob}[M_0(x) \neq M_1(x) : (M_0, M_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda), \; x \leftarrow \mathcal{A}(1^\lambda, M_0, M_1, \mathsf{aux})] = negl(\lambda)$$

Remark 1. Note that for simplicity we have assumed, that all Turing machines in family $\mathcal{M}$ outputs the time τ_x in addition to the output on input x. The above definition in particular implies that there does not exist any efficient adversary who can produce x such that the two Turing machines output by the sampler on x run in different times. We stress that this has been done for simplicity and our definition can also deal with a family of machines with different running times by just padding.

Similar to the case of circuits, we define the notion of differing-inputs obfuscation for a differing-inputs family of Turing machines.

Definition 4. ***(Differing-inputs Obfuscators for Turing machines)** A uniform PPT machine $\mathsf{di}\mathcal{O}$ is called a Turing machine differing-inputs Obfuscator defined for differing-inputs Turing machine family $\mathcal{M}$, if the following conditions are satisfied:*

- ***Correctness**: For all security parameters $\lambda \in \mathbb{N}$, for all $M \in \mathcal{M}$, for all inputs x, we have that*

$$\mathsf{Prob}[M'(x) = M(x) \; : \; M' \leftarrow \mathsf{di}\mathcal{O}(\lambda, M)] \; = \; 1$$

- ***Differing-inputs obfuscation property**: For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for $(M_0, M_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}_{\mathcal{M}}(1^\lambda)$ we have that*

$$|\mathsf{Prob}[D(\mathsf{di}\mathcal{O}(\lambda, M_0), \mathsf{aux}) = 1] \; - \; \mathsf{Prob}[D(\mathsf{di}\mathcal{O}(\lambda, M_1), \mathsf{aux}) = 1] \leq \alpha(\lambda)$$

*In addition to the above properties if $\mathsf{di}\mathcal{O}$ satisfies the following properties, with respect to a universal polynomial p, then we say that $\mathsf{di}\mathcal{O}$ is **succinct** and has **input-specific run time**. Let $M' \leftarrow \mathsf{di}\mathcal{O}(\lambda, M)$.*

- ***Succinct****: The size of M' is $p(\lambda, |M|)$, where $|M|$ denotes the size of the Turing machine M.*
- ***Input-specific run time****: The running time of M' on an input x is $p(\lambda, \mathsf{time}(M, x))$.*

Remark 2. Our definition of differing-inputs obfuscation just says that the definition can be with respect to a class of admissible samplers, instead of with respect to all possible samplers. This would allow us to circumvent all known trade-off results for differing-inputs obfuscation including the recent results by Boyle et al. [BP13] and Garg et al. [GGHW13] (note there are no impossibility results), which are only with respect to specific auxiliary information. Our applications would then hold with respect to a particular sampler outputting specific auxiliary distribution, if the assumption held for a different class of samplers, possibly, associated to a different auxiliary distribution.

We can also consider the notion of indistinguishability obfuscation for Turing machines. The definition is very similar to the above definition except that the indistinguishability of obfuscations holds only only for Turing machines which are same on all inputs. We present the formal definition in Appendix A for the sake of completeness. We note that our construction of Turing machine differing-inputs obfuscation also satisfies the definition of Turing machine indistinguishable obfuscation since Definition 4 implies Definition 6.

3 Differing-inputs Obfuscators for Turing Machines

In this section, we construct differing-inputs obfuscators for Turing machines. The advantage of considering obfuscation of Turing machines over circuits is two-fold. Firstly, the running time of the obfuscated Turing machine would be input specific. Secondly, the size of the obfuscation does not depend on the worst case running time of the Turing machine. Since the real world applications are programs it is more natural to consider obfuscation of Turing machines rather than circuits.

Our construction is based on the assumption that the differing-input obfuscator for all circuits exists along with well studied assumptions such as the existence of FHE, SNARKs and collision resilient hash functions.

3.1 Tools

We now describe the main cryptographic tools that we use in our construction.

Universal Turing machines. Universal Turing machine takes as input a Turing machine, an input on which the Turing machine is executed and a time to indicate the number of steps of execution. The output of the univeral Turing machine is basically the output of the Turing machine on that input if the execution is completed within the time limit, which is given as input to the universal Turing machine. Otherwise, the universal Turing machine outputs $\perp$. We consider a variant of the universal Turing machine, that instead of outputting the entire result of execution, will just output one particular bit from the result of execution. More formally, we define the variant as follows. For every $1 \leq i \leq t$, represent by $\mathsf{UTM}^{(i)}_{y,t}(\cdot)$, the following program: It takes as input a Turing machine M' and executes M' on y for t steps. If the execution is completed within t steps then output the i^{th} bit of the output of the execution otherwise output $\perp$.

FHE for Turing machines. Goldwasser et al. in [GKP+13a] build a compiler that takes a Turing machine M along with the number of steps t as input and then produces a Turing machine that computes the FHE evaluation of M for t number of steps. In more detail, the compiler

converts the machine into an oblivious Turing machine M using the Pippenger-Fischer [PF79] transformation. It then constructs a new Turing machine M_{FHE} which takes a ciphertext along with a FHE public key as input and executes the oblivious Turing machine fully homomorphically on the ciphertext for t number of steps. The output of the compiler is M_{FHE}. The compiler, denoted by $\mathsf{Compile}^{\mathsf{TM}}_{\mathsf{FHE}}$, is described formally in Appendix B.2.

SNARKs. Succinct non interactive arguments of knowledge, referred to as SNARKs, are arguments where the proof sent by the prover to the verifer is succinct. By succinct we mean that the size of the proof is upper bounded by a fixed polynomial in the security parameter and is independent of the instance for which the proof is given. Further, SNARK verifier runs in time that depends only on the size of the input instance and the security parameter, and not on the size of the witness. In addition to these properties, SNARKs also satisfy the property of extractability – there exists an extractor that given a trapdoor and a convincing proof, can extract a witness used to generate the proof. SNARKs have been constructed under knowledge assumptions [BCCT13]. The formal details of SNARKs are presented in Appendix B.3. We denote the SNARK proof system we use by (Setup, P, V).

We occasionally refer to a weaker notion of SNARKs, referred to as SNARGs (Succinct Non-interactive Arguments of Knowledge) [BCCT13, GW11]. In place of the extractability property, SNARGs have the weaker property of soundness – there does not exist any efficient dishonest prover who can convince a verifier with non-negligible probability that a false statement belongs to a language.

Hash functions. The final tool we require for our construction are collision-resilient hash functions that map arbitrary length input to a fixed length output. More formally, we consider a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{l(\lambda)}$, where l is a polynomial [5]. There are constructions of such functions known in the literature [Mer90, Dam90, GK03]. Henceforth, we refer to such functions as collision-resilient size reducing hash functions.

3.2 Construction

We are now ready to describe the construction. Before we do this, we first describe a class of programs $\mathcal{P}$, represented by a circuit family to which we apply differing-inputs obfuscation for circuits. Each program in this class is indexed by $(g_1, g_2, \mathsf{CRS}, \mathsf{PK}_1, \mathsf{PK}_2)$. We denote such a program by $\mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}_1,\mathsf{PK}_1,\mathsf{PK}_2)}$. Here, $(\mathsf{PK}_1, \mathsf{SK}_1), (\mathsf{PK}_2, \mathsf{SK}_2)$ denotes the FHE public key-secret key pairs, g_1, g_2 denote the encryptions of M with respect to PK_1 and PK_2 respectively and CRS denotes the common reference string output by the SNARK setup algorithm. We describe $\mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}_1,\mathsf{PK}_1,\mathsf{PK}_2)}$ in Figure 1.
We are now ready to give the construction of differing-inputs obfuscation of a differing-input Turing machine family denoted by $\mathcal{M}$.

The obfuscation of a Turing machine is captured by the obfuscate algorithm, denoted by $\mathsf{Obfuscate}_{\mathsf{TM}}$. As in Garg et. al. [GGH+13b], we view the execution of the obfuscated Turing machine on an input as an evaluation algorithm, denoted by $\mathsf{Evaluate}_{\mathsf{TM}}$.

$\mathsf{Obfuscate}_{\mathsf{TM}}(1^\lambda, M)$: The obfuscation algorithm on input a security parameter and a Turing machine $M \in \mathcal{M}$ does the following:

[5] More formally, we do the following. We consider a hash function family from which we sample a hash function $\mathcal{H}$. Whenever we use $\mathcal{H}$, we implicitly mean that $\mathcal{H}$ was sampled from an appropriate distribution on the hash function family.

$$\mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}_1,\mathsf{PK}_1,\mathsf{PK}_2)}$$

Given input $(i, e_1^{(i)}, e_2^{(i)}, h_x, t, \varphi)$, $\mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK},\mathsf{PK})}$ proceeds as follows:

1. Execute the SNARK verifier V on input $(e_1^{(i)}, e_2^{(i)}, g_1, g_2, h_x, t, i)$ along with CRS and proof φ. The SNARK proof system (P, V) is defined for the language L where $(e_1^{(i)}, e_2^{(i)}, g_1, g_2, h_x, t, i)$ are instances in L with witnesses x such that:

$$M^{(i)}_{\mathsf{FHE}} = \mathsf{Compile}^{\mathsf{TM}}_{\mathsf{FHE}}(\mathsf{UTM}^{(i)}_{x,2^t}, 2^t\mathsf{log}2^t) \text{ and } e_1^{(i)} = M^{(i)}_{\mathsf{FHE}}(\mathsf{PK}_1, g_1)$$
$$\text{and } e_2^{(i)} = M^{(i)}_{\mathsf{FHE}}(\mathsf{PK}_2, g_2) \text{ and } \mathcal{H}(x) = h_x \quad (1)$$

2. If the verifier rejects then output 0; otherwise, output $\mathsf{Decrypt}_{\mathsf{FHE}}(e_1^{(i)}, \mathsf{SK}_1)$.

Figure 1 A template of a program in the program class $\mathcal{P}$.

1. Generate $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{SK}^1_{\mathsf{FHE}}) \leftarrow \mathsf{Setup}_{\mathsf{FHE}}(1^\lambda)$ and $(\mathsf{PK}^2_{\mathsf{FHE}}, \mathsf{SK}^2_{\mathsf{FHE}}) \leftarrow \mathsf{Setup}_{\mathsf{FHE}}(1^\lambda)$.
2. Generate ciphertexts $g_1 = \mathsf{Encrypt}_{\mathsf{FHE}}(\mathsf{PK}^1_{\mathsf{FHE}}, M)$ and $g_2 = \mathsf{Encrypt}_{\mathsf{FHE}}(\mathsf{PK}^2_{\mathsf{FHE}}, M)$.
3. Compute CRS by executing the setup algorithm Setup corresponding to the SNARK proof system, denoted by (Setup, P, V) for the relation described in Equation 1 in Figure 1.
4. Generate a differing-inputs obfuscation for the circuit $\mathsf{P1} = \mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^1_{\mathsf{FHE}},\mathsf{PK}^1_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})}$ as $\mathsf{P1}_{\mathsf{obf}} = \mathsf{di}\mathcal{O}(\mathsf{P1}, \lambda)$.
5. The output of this algorithm is $\sigma = (\mathsf{P1}_{\mathsf{obf}}, \mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2, \mathsf{CRS})$.

$\mathsf{Evaluate}_{\mathsf{TM}}(\sigma = (\mathsf{P1}_{\mathsf{obf}}, \mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2, \mathsf{CRS}), x)$: On input the obfuscation of a Turing machine M and input x, the $\mathsf{Evaluate}_{\mathsf{TM}}$ algorithm outputs $M(x)$ as follows.

1. Compute the hash of x using the hash function, $\mathcal{H}$ and denote the result by h_x $(= \mathcal{H}(x))$.
2. Repeat the following for steps $t = 0, 1, 2, \ldots$:
   - Execute $\mathsf{Compile}^{\mathsf{TM}}_{\mathsf{FHE}}(\mathsf{UTM}^{(i)}_{x,2^t},\ 2^t\mathsf{log}2^t)$ [6], for all $1 \le i \le 2^t$, to obtain $M^{(i,t)}_{\mathsf{FHE}}$ [7].
   - Compute $e_1^{(i,t)} = M^{(i,t)}_{\mathsf{FHE}}(\mathsf{PK}^1_{\mathsf{FHE}}, g_1)$ and $e_2^{(i,t)} = M^{(i,t)}_{\mathsf{FHE}}(\mathsf{PK}^2_{\mathsf{FHE}}, g_2)$, where $1 \le i \le 2^t$.
   - For every $1 \le i \le 2^t$, compute SNARK proof φ_i, using prover P, that the encryptions $e_1^{(i,t)}$ and $e_2^{(i,t)}$ as well as the hash value h_x are computed correctly as in Equation 1 in Figure 1.
   - For every $1 \le i \le 2^t$, run $\mathsf{P1}_{\mathsf{obf}}(i, e_1^{(i,t)}, e_2^{(i,t)}, h_x, t, \varphi_i)$. If the output of the program is $\perp$ [8] then go to the beginning of the loop. Else, first assign b_i to be the output of $\mathsf{P1}_{\mathsf{obf}}$. Consider the concatenation of b_i, for $1 \le i \le 2^t$ to be out. The output of $\mathsf{Evaluate}_{\mathsf{TM}}$ is out.

[6] A universal Turing machine that executes an input Turing machine for T steps, itself takes $O(T\mathsf{log}T)$ number of steps.

[7] Note that we need to execute the compile algorithm for every output bit. But, we know that the output length of a Turing machine cannot exceed the running time required to produce that output. And so, we execute the compile algorithm for 2^t number of steps.

[8] Here we assume that the output tape of the Turing machine contains $\perp$ until the execution of the Turing machine is completed. After the execution is completed, the $\perp$ symbol is replaced by the output of the execution.

Remark: The construction as described above relies on the existence of SNARKs. Later, in Lemma 1 stated in Appendix C, we will see that we need SNARKs because we need to extract the witness x corresponding to the NP-statement in Equation 1 described in Figure 1.

Alternatively, differing inputs obfuscation assuming just SNARGs can be achieved if the inputs to the Turing machine are upper-bounded by some fixed parameter as follows. Instead of passing h_x to the program in Figure 1, we can directly pass x to the program. Note that this could not be done if the input length was not apriori bounded because the input length of the circuit implementing the program in Figure 1 is fixed. Since we are directly including x as part of the input, we can use SNARGs instead of SNARKs since the whole purpose of using SNARKs was to obtain x.

We first show that the construction satisfies both the succinctness as well as the input-specific running time properties. The proof of correctness as well as proof of security is presented in Appendix C. Using $\mathsf{di}\mathcal{O}$ for Turing machines, we then construct a functional encryption scheme where the secret keys are succinct and the decryption time is input-specific. The details are presented in Appendix D. We also present a delegatable functional encryption scheme in Appendix E.

Size of Obfuscation: We now upper bound the size of the obfuscated Turing machine which is obtained by feeding as input, Turing machine M, to the $\mathsf{Obfuscate}_{\mathsf{TM}}$ algorithm. Denote the output of the $\mathsf{Obfuscate}_{\mathsf{TM}}$ algorithm to be $(\mathsf{P1}_{\mathsf{obf}}, \mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2, \mathsf{CRS})$.

- The size of the FHE public keys $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}})$ can be upper bounded by a polynomial in the security parameter.
- The size of the FHE encryptions g_1 and g_2 depends on the size of the message, which is in this case, Turing machine M along with the security parameter. That is, the size of g_1 and g_2 can be upper bounded by a polynomial in $(\lambda, |M|)$.
- The size of the common reference string CRS is a polynomial in the security parameter (refer to Appendix B.3).
- The size of the obfuscation $\mathsf{P1}_{\mathsf{obf}}$ is a polynomial in $(\lambda, |\mathsf{P1}|)$, where $|\mathsf{P1}|$ represents the size of the circuit of $\mathsf{P1}$. We now argue about the size of $\mathsf{P1}$. Program $\mathsf{P1}$ consists of two main components – the SNARK verifier circuit and the decryption circuit. The size of the SNARK verifier circuit is a polynomial in its inputs. The inputs to the SNARK verifier circuit consists of the following.
  - A pair of encryptions of Turing machine M. The size of this is a polynomial in $(\lambda, |M|)$.
  - Encryptions of a single bit of the output of the Turing machine. The size of this is just a polynomial in λ.
  - Index of the output bit. The size of this is logarithmic in the running time of the Turing machine. Since we are only interested with efficient Turing machines, we can loosely upper bound this quantity by the security parameter λ.
  - Iteration number t. Observe that this too can be can be loosely upper bounded by the security parameter λ.

  The decryption circuit on the other hand takes a ciphertext corresponding to the encryption of a single bit and hence, the size of the decryption circuit is a polynomial in the security parameter. Combining all the facts, we get $|\mathsf{P1}|$ to be a polynomial in $(\lambda, |M|)$.

Running time of Evaluate algorithm: We first make the following observation. Suppose the Turing machine M takes time T to execute on input x then the number of iterations in the evaluation procedure need to be performed is $O(T\mathsf{log}T)$. We then determine the amount of time taken in each iteration t step by step.

- In the first step, the FHE compiler is executed which takes time which is a polynomial in 2^t, and hence at most time T, and the size of the universal Turing machine $\mathsf{UTM}^i_{(x,2^t)}$ (Refer Appendix B.2). Further, the size of the universal Turing machine is a polynomial in $|x|$ and the security parameter λ. Hence, the running time of the compiler is essentially a polynomial in $(\lambda, T, |x|)$.
- In the second step, $M^{(i,t)}_{\mathsf{FHE}}$ is executed twice and the running time of this step is a polynomial in (T, λ).
- In the third step, a SNARK proof is computed and the running time of this is nothing but the running time of the SNARK prover. The running time of the SNARK prover is a polynomial in $(\lambda, |M|, |x|, T)$.
- In the fourth step, the obfuscation algorithm is evaluated. Since this obfuscation is in the form of a circuit, the running time of this depends on the size of the obfuscation, which is nothing but a polynomial in $(\lambda, |M|)$.

Finally, the running time of computing hash on input x is a polynomial in $(|x|, \lambda)$. From the above points, we have that the running time of the Evaluate algorithm is a polynomial in $(\lambda, |M|, x, T)$.

4 Multiparty Key Exchange and Broadcast Encryption with Small Parameters

In this section, we build multiparty non-interactive key exchange (NIKE) and broadcast encryption from differing-inputs obfuscation. Our constructions can be built from any differing-inputs obfuscator and any collision-resistant hash function.

First, we review Merkle hash trees and puncturable pseudorandom functions (PRFs). Given a collision-resistant hash function $H : \mathcal{X}^2 \to \mathcal{X}$, a Merkle hash tree [Mer88] gives another collision-resistant hash function $\mathcal{H} : \mathcal{X}^n \to \mathcal{X}$ where $n = 2^k$ for some fixed k. The input consists of 2^k blocks $x_k[i] \in \mathcal{X}$ for $i \in \{1, \ldots, 2^k\}$. These blocks are set as the leaves of a binary tree with 2^k leaf nodes. The value at each internal node is obtained by hashing the values of that node's children. The output of $\mathcal{H}$ is the value at the root of the tree. More precisely, for each $j \in \{0, \ldots, 2^{k-1} - 1\}$, blocks $x_k[2j]$ and $x_k[2j + 1]$ are hashed using H to obtain $x_{k-1}[j] = H(x_k[2j], x_k[2j + 1])$. This process is repeated for $k - 1, k - 2, \ldots, 1$ until a single block $x_0 \in \mathcal{X}$ is obtained. The output of $\mathcal{H}$ is set to x_0.

We need the following standard property of Merkle Hash Trees. Let $y \in \mathcal{X}$ and $x \in \mathcal{X}^n$ such that $x[i] = y$ for some i. Since $\mathcal{H}$ is collision resistant we can treat $h = \mathcal{H}(x)$ as a binding commitment to x. The property we need is that, given h and y, it is possible to produce a short proof that $x[i] = y$. The proof consists of $x_k[i]$ and the values at all siblings of nodes on the path from $x_k[i]$ to the root of the Merkle tree. The size of a proof is $O(\log n)$ elements of $\mathcal{X}$. False proofs exist, but they lead directly to a collision for H. These proofs can be generalized to the case where y consists of p (not necessarily contiguous) blocks, and the size of proof will be $O(p \log n)$ elements of $\mathcal{X}$.

Following [BW13, BGI13, KPTZ13], a puncturable pseudorandom function F is a pseudorandom function (PRF) that supports the a procedure $F^x \leftarrow F.\mathsf{Puncture}(x)$ where

$$F^x(y) = \begin{cases} F(y) & \text{if } y \neq x \\ \perp & \text{if } y = x \end{cases}$$

For security, we let an adversary $\mathcal{A}$ commit to a point x. $\mathcal{A}$ receives F^x, as well as a value z, where either $z = F(x)$ or z is chosen uniformly in the codomain of F. A puncturable PRF is

secure if no efficient adversary A can distinguish the correct z from a random z. We note that the PRF construction of Goldreich, Goldwasser, and Micali [GGM86] satisfies this functionality and notion of security.

4.1 Non-interactive Multiparty Key Exchange

A NIKE protocol consists of the following three algorithms:

- $\mathsf{Setup}(\lambda, n)$: The setup algorithm takes a security parameter λ and a number n of users. It outputs public parameters PP.
- $\mathsf{Publish}(\mathsf{PP}, i)$: Each party executes the publishing algorithm, which takes as input the public parameters and the index of the party, and generates two values: a user secret key SK_i and a user public value PV_i. User i keeps SK_i as his secret, and publishes PV_i to the other users.
- $\mathsf{KeyGen}(\mathsf{PP}, i, \mathsf{SK}_i, \{\mathsf{PV}_j\}_{j=1,\ldots,n})$: Finally, each party derives the shared key k using the public parameters PP, their secret SK_i, and the other parties' public values $\{\mathsf{PV}_j\}_{j=1,\ldots,n}$.

Static security for a NIKE protocol is defined by the following experiment denoted by $\texttt{EXP}(b)$ and parameterized by the total number of parties n and a bit $b \in \{0,1\}$ on an adversary $\mathcal{A}$:

$\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$

$(\mathsf{SK}_i, \mathsf{PV}_i) \leftarrow \mathsf{Publish}(1^\lambda, i)$ for $i = 1, \ldots n$

$b' \leftarrow \mathcal{A}\big(\ \mathsf{PP},\ \{\mathsf{PV}_i\}_{i=1,\ldots,n},\ k^*\ \big)$
where
$k_0 \leftarrow \mathsf{KeyGen}(\mathsf{PP}, \{\mathsf{PV}_i\}_{i=1,\ldots,n}, \mathsf{SK}_1, 1)$, $k_1 \leftarrow \{0,1\}^\lambda$, and $k^* \leftarrow k_b$

For $b = 0, 1$ let W_b be the event that $b' = 1$ in $\texttt{EXP}(b)$ and define $\texttt{AdvKE}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 5. *A multiparty key exchange protocol* $(\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ *is statically secure if, for any PPT adversary* $\mathcal{A}$ *and any integer* n*, the function* $\texttt{AdvKE}(\lambda)$ *is negligible.*

Construction Let F be a puncturable pseudorandom function, $f : \mathcal{X} \to \mathcal{Y}$ a one-way function, and $\mathcal{H} : \mathcal{Y}^n \to \mathcal{Y}$ a Merkle Hash Tree.

- $\mathsf{Setup}_{\mathsf{NIKE}}(1^\lambda, 1^n)$: The $\mathsf{Setup}_{\mathsf{NIKE}}$ algorithm takes the security parameter λ and a number of users n and computes the following:
  1. Generate an instance F of a puncturable pseudorandom function with security parameter λ.
  2. Compute the differing-inputs obfuscation $\mathsf{P1}_{\mathsf{obf}}$ of the program $\mathsf{P1} = \mathsf{P}^{(F)}$ from Figure 2, using the size of the circuit to be $\max\{|\mathsf{P}^{(F)}|, |\mathsf{P}_2^{(h^*, F^{h^*})}|\}$ where $\mathsf{P}_2^{(h^*, F^{h^*})}$ is defined in Figure 3 in Appendix F.

  It sets the public parameters as
  $$\mathsf{PP} = \mathsf{P1}_{\mathsf{obf}}$$
- $\mathsf{Publish}_{\mathsf{NIKE}}(1^\lambda, i)$: User i chooses a random $x_i \in \mathcal{X}$, and computes $y_i = f(x_i) \in \mathcal{Y}$. User i keeps x_i as its secret key, and publishes y_i as its public value.
- $\mathsf{KeyGen}_{\mathsf{NIKE}}(\mathsf{PP}, \{y_j\}_{j=1,\ldots,n}, x_i, i)$: To compute the shared secret, user i computes the Merkle hash $h = \mathcal{H}(y_1, \ldots, y_n)$, and constructs a proof π that it knows a z such that $\mathcal{H}(z) = h$ and $z[i] = y_i$. Then it computes $k \leftarrow \mathsf{P1}_{\mathsf{obf}}(h, \pi, i, y_i, x_i)$.

Correctness. Correctness of our scheme is straightforward by inspection.

$\mathsf{P}^{(F)}$

Given input (h, π, i, y, x), $\mathsf{P}^{(F)}$ proceeds as follows:

1. Check that π is a valid proof that there exists $z \in \mathcal{Y}^n$ where $\mathcal{H}(z) = h$ and $z[i] = y$.
2. Check that $f(x) = y$.
3. If either check fails, abort and output $\perp$.
4. Otherwise, output $F(h)$

Figure 2 The program $\mathsf{P}^{(F)}$ that users will use for key generation.

Parameter sizes. Secret keys in our scheme just elements in the domain of a one-way function, which is independent of the number of users. published values are images, which are also independent of the number of users. The public key is an obfuscation of the program in Figure 2, which only depend logarithmically on the number of users.

Untrusted Setup. As described, our key exchange requires a trusted setup. However, as in [BZ13], $\mathsf{Setup}_{\mathsf{NIKE}}$ can be run independently from $\mathsf{Publish}_{\mathsf{NIKE}}$. Therefore, we can set party 1 as the "master party" who runs $\mathsf{Setup}_{\mathsf{NIKE}}$ in addition to $\mathsf{Publish}_{\mathsf{NIKE}}$, and publishes the public key along with his published value. We note that the material published by player 1 is still relatively small: polylogarithmic in the number of users. This is in contrast to the scheme of [BZ13], where player 1 must publish material of size *polynomial* in the number of users.

Security. The security of our scheme is given by the following theorem:

Theorem 1. *The scheme above is statically secure if $\mathcal{H}$ is a collision resistant Merkle hash tree, F is a secure punctured PRF, f is a secure one-way function, and the $\mathsf{P1}_{\mathsf{obf}}$ is a differing-input obfuscation of $\mathsf{P}^{(F)}$.*

Here we sketch the proof — the full proof appears in Appendix F. Let h^* be the the hash of the published values $y_1, \ldots, y_n$. We puncture F at h^*, and add a check to the program $\mathsf{P}^{(F)}$ that $h \neq h^*$, resulting in a program $\mathsf{P}_2^{(h^*, F^{h^*})}$. The only inputs where $\mathsf{P}^{(F)}$ and $\mathsf{P}^{(h^*, F^{h^*})}$ differ have the form (h^*, π, i, y, x) where $f(x) = y$ and π is a valid proof that there exists $z \in \mathcal{Y}^n$ where $\mathcal{H}(z) = h^*$ and $z[i] = y$. We argue that such inputs are hard to compute. There are two cases:

- $y = y_i$. Then x is a pre-image of y_i under f. We can use such an input to break the one-wayness of f.
- $y \neq y_i$. Then since π is valid, but $h^* = \mathcal{H}(y_1, \ldots, y_n)$ with $y_i \neq y$, the proof must yield a collision on the underlying collision resistant hash H.

Therefore, the only inputs on $\mathsf{P}^{(F)}$ and $\mathsf{P}_2^{(h^*, F^{h^*})}$ differ are hard to compute. Differing inputs obfuscation thus implies that the obfuscations of the two programs are indistinguishable. At this point, the adversary's view only depends on the punctured PRF F^{h^*}, and the security of F shows that the shared secret $k = F(h^*)$ is therefore indistinguishable from random, as desired.

4.2 Broadcast Encryption

Boneh and Zhandry [BZ13] give a generic conversion from a multiparty key exchange protocol into a broadcast encryption scheme. Applying to our key exchange protocol above, we obtain a broadcast scheme with compact ciphertexts and secret keys (namely, independent of the number

of users). However, the encryption key resulting from this conversion contains a public value for every user, and is therefore linear in size. In Appendix F, we give a direct construction using many of the same ideas as above. Our construction maintains constant-sized ciphertexts and secret keys while shrinking the public encryption key to logarithmic in the number of users.

References

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 111–120. ACM, 2013.

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

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, pages 103–112, 1988.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. Cryptology ePrint Archive, Report 2013/401, 2013.

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. Cryptology ePrint Archive, Report 2013/631, 2013. `http://eprint.iacr.org/`.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. *Advances in Cryptology — CRYPTO 2005*, pages 1–19, 2005.

[BP13] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. Cryptology ePrint Archive, Report 2013/703, 2013. `http://eprint.iacr.org/`.

[BR13] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. `http://eprint.iacr.org/`.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.

[BT13] Mihir Bellare and Stefano Tessaro. Poly-many hardcore bits for any one-way function. Cryptology ePrint Archive, Report 2013/873, 2013. `http://eprint.iacr.org/`.

[BW13] Dan Boneh and Brent Waters. Constrained Pseudorandom Functions and Their Applications. *Advances in Cryptology — AsiaCrypt 2013*, pages 1–23, 2013.

[BZ13] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/642, 2013. `http://eprint.iacr.org/`.

[Dam90] Ivan Bjerre Damgård. A design principle for hash functions. In *Advances in Cryptology—CRYPTO'89 Proceedings*, pages 416–427. Springer, 1990.

[Del07] Cécile Delerablée. Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys. 2:200–215, 2007.

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. *PAIRING 2007*, (July), 2007.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. *Advances in Cryptology — CRYPTO 1993*, 773:480–491, 1994.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *Proc. of FOCS (to appear)*, 2013.

[GGHW13] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. Cryptology ePrint Archive, Report 2013/860, 2013. `http://eprint.iacr.org/`.

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

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in) security of the fiat-shamir paradigm. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 102–113. IEEE, 2003.

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

[GKP+13b] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. In *STOC*, 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 99–108. ACM, 2011.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. pages 443–457, 2000.

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

[LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious ram for secure two-party computation. In *TCC*, pages 377–396, 2013.

[Mer88] Ralph Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1988.

[Mer90] Ralph C Merkle. One way hash functions and des. In *Advances in Cryptology—CRYPTO'89 Proceedings*, pages 428–446. Springer, 1990.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). pages 294–303, 1997.

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

[PPS13] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. Cryptology ePrint Archive, Report 2013/754, 2013. `http://eprint.iacr.org/`.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[SF07] Ryuichi Sakai and Jun Furukawa. Identity-Based Broadcast Encryption. *IACR Cryptology ePrint Archive*, 2007.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *CCS*, pages 463–472, 2010.

[SW08] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In *Automata, Languages and Programming*, pages 560–578. Springer, 2008.

A Indistinguishability Obfuscation for Turing machines and FE for Turing machines: Definitions

A.1 $\mathsf{i}\mathcal{O}$ for Turing machines

We define the notion of indistinguishability obfuscation for Turing machines similar to the definition of indistinguishability obfuscation for circuits. Note that the two Turing machines that need to be obfuscated in the security game, should not only be identical on all inputs but they also need to having the same running time on all the inputs. Note that the definition of differing-inputs obfuscation for Turing machines implies the following definition and hence our construction in Section 3 also satisfies this definition.

Definition 6. ***(Indistinguishable Obfuscators for Turing machines)*** *A uniform PPT machine $\mathsf{i}\mathcal{O}_{\mathsf{TM}}$ is called a Turing machine indistinguishable Obfuscators for the Turing machine family $\mathcal{M}$, if the following conditions are satisfied:*

- ***Correctness**: For all security parameters $\lambda \in \mathbb{N}$, for all $M \in \mathcal{M}$, for all inputs x, we have that*
$$\mathsf{Prob}[M'(x) = M(x) \ : \ M' \leftarrow \mathsf{i}\mathcal{O}_{\mathsf{TM}}(\lambda, M)] \ = \ 1$$
- ***Indistinguishable Obfuscation**: For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all $M_0, M_1 \in \mathcal{M}, \mathsf{aux}$ such that for all x, $M_0(x) = M_1(x)$ and $\mathsf{time}(M_0, x) = \mathsf{time}(M_1, x)$ we have that*
$$|\mathsf{Prob}[D(\mathsf{i}\mathcal{O}_{\mathsf{TM}}(\lambda, M_0)) = 1] \ - \ \mathsf{Prob}[D(\mathsf{i}\mathcal{O}_{\mathsf{TM}}(\lambda, M_1)) = 1] \leq \alpha(\lambda)$$

*In addition to the above properties if $\mathsf{i}\mathcal{O}_{\mathsf{TM}}$ satisfies the following properties, with respect to a universal polynomial p, then we say that $\mathsf{i}\mathcal{O}_{\mathsf{TM}}$ is **succinct** and has **input-specific run time**.*

- ***Succinct**: The size of M' is $p(\lambda, |M|)$, where $|M|$ denotes the size of the Turing machine M.*
- ***Input-specific run time**: The running time of M' on an input x is $p(\lambda, \mathsf{time}(M, x))$.*

A.2 Functional Encryption for Turing machines

We cast the definition of functional encryption in Garg et al. [GGH+13b] for the case of Turing machines. Though the notion of functional encryption for Turing machines have already been defined by Goldwasser et al. [GKP+13a], our definition differs from their definition in many ways – (i) single key versus many key queries, (ii) simulation based versus indistinguishability game based and so on. While defining FE for Turing machines we restrict the adversary to make only certain type of function queries which is based on the running time of the function. This is to avoid trivial attacks where the attacker will be able to distinguish the encryptions of two messages by choosing a function query whose running time is significantly different on both the messages.

Definition 7. *Let the message space be $\mathcal{S} = \mathcal{S}_\lambda$. A functional encryption scheme defined for a family of Turing machines $\mathcal{M}_\mathcal{T}$, parameterized by a Turing machine $\mathcal{T}$, consists of four algorithms* $\mathsf{FE} = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}\}$*:*

- $\mathsf{Setup}(1^\lambda)$ *- a polynomial time algorithm that takes the unitary representation of the security paramter λ and outputs a public paramteres* PP *and a master secret key* MSK*.*
- $\mathsf{KeyGen}(\mathsf{MSK}, f)$ *- a polynomial time algorithm that takes as input the master secret key* MSK *and a function f implementable by a Turing machine $M \in \mathcal{M}$ and outputs a corresponding secret key* SK_f*.*
- $\mathsf{Encrypt}(\mathsf{PP}, x)$ *- a polynomial time algorithm that takes the public parameters* PP *and a string $x \in \mathcal{S}$ and outputs a ciphertext* CT*.*
- $\mathsf{Decrypt}(\mathsf{SK}_f, \mathsf{CT})$ *- a polynomial time algorithm that takes a secret key* SK_f *and ciphertext encrypting message $x \in \mathcal{S}$ and outputs $f(x)$.*

A functional encryption scheme is correct for $\mathcal{M}$ if for all $M \in \mathcal{M}$ and all messages $x \in \mathcal{S}$:

$$\mathsf{Prob}[(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{Decrypt}(\mathsf{KeyGen}(\mathsf{MSK}, f), \mathsf{Encrypt}(\mathsf{PK}, x)) = f(x)] = \mathsf{negl}(\lambda)$$

We now define the (fully) indistinguishability security for functional encryption which is described in form of an indistinguishability game between an attacker $\mathcal{A}$ and a challenger.

Setup: The challenger runs $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives PP to $\mathcal{A}$.

Query: $\mathcal{A}$ submits queries $f_i \in \mathcal{M}$. We assume, without loss of generality that f_i can be represented by a Turing machine M_i which, on an input x, outputs $f_i(x)$ along with the time taken by M_i to execute on x. The adversary $\mathcal{A}$ is then given $\mathsf{SK} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f_i)$

Challenge: The adversary then outputs messages (x_0, x_1) such that $f(x_0) = f(x_1)$ for all f submitted by $\mathcal{A}$ during the query phase.

Query: $\mathcal{A}$ executes another query phase. It submits queries of the form $f_i \in \mathcal{M}$ which are represented by Turing machines M_i as in the previous Query phase. If $f_i(x_0) = f_i(x_1)$, adversary $\mathcal{A}$ is given $\mathsf{SK} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f_i)$ else the game is aborted.

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

The advantage of an adversary $\mathcal{A}$ is defined to be $|\mathsf{Prob}[b' = b] - \frac{1}{2}|$.

Definition 8. *A functional encryption scheme is (fully) indistinguishability secure if for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above indistinguishability game is negligible.*

In addition to the above properties, we also consider the following two properties for functional encryption schemes for Turing machines.

- **Succinctness**: A functional encryption scheme is said to be succinct if the functional key generated using KeyGen for the function f is $p(\lambda, |M|)$, where p is a polynomial and M denotes the size of the Turing machine representing the function f.
- **Input-specific run time**: A functional encryption scheme is said to have input-specific run time if the decryption algorithm on input a functional key for a function f along with an encryption of x, takes time $p(\lambda, \mathsf{time}(M, x))$, where M is the Turing machine representing the function f.

A weaker notion, called *selective security* can also be considered where the adversary submits his challenge message to the challenger even before the challenger executes the setup phase.

B Background

B.1 Fully Homomorphic Encryption

We define the notion of fully homomorphic encryption (FHE) scheme. It consists of four PPT algorithms $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Eval})$ defined as follows.

- $\mathsf{KeyGen}(1^\lambda)$: On input a security parameter 1^λ it outputs a public key $\mathsf{PK_{FHE}}$ and a decryption key $\mathsf{SK_{FHE}}$.
- $\mathsf{Encrypt}(m, \mathsf{PK_{FHE}})$: On input a message m and public key $\mathsf{PK_{FHE}}$ it outputs a ciphertext denoted by CT.
- $\mathsf{Decrypt}(\mathsf{CT}, \mathsf{SK_{FHE}})$: On input a ciphertext and a decryption key $\mathsf{SK_{FHE}}$ it outputs a message m.
- $\mathsf{Eval}(\mathsf{CT}, \mathsf{PK_{FHE}}, f)$: On input a ciphertext, a public key [9] and a function f, outputs another ciphertext CT' such that the decryption of CT' yields the message $f(m)$.

The security of FHE is defined very similar to the security of the IND-CPA public key encryption scheme. There does not exist any PPT adversary A such that, for any pair of messages m_0, m_1, the probability that on input $\mathsf{Encrypt}(m_0, \mathsf{PK_{FHE}})$ it outputs 0 is negligibly close to the probability that on input $\mathsf{Encrypt}(m_1, \mathsf{PK_{FHE}})$ it outputs 0.

B.2 FHE for Turing machines

We present the construction of the compiler verbatim from Goldwasser et. al. [GKP+13a] below. The compiler, denoted by $\mathsf{Compile}^{\mathsf{TM}}_{\mathsf{FHE}}$, takes as input a Turing machine M and a number of steps t, and produces a Turing machine that computes the FHE evaluation of M for t steps. Let $\widehat{x}$ denote the FHE encryption of x.

$\mathsf{Compile}^{\mathsf{TM}}_{\mathsf{FHE}}(M, t)$:

- First, transform M into an oblivious Turing machine M_O by applying the Pippenger-Fischer transformation [PF79] for time bound t. This transformation results in a new Turing machine M_O and a transition function δ for M_O. Namely, δ takes as input tape input bit b, a state state and outputs a new state state', new content b' for the tape location, and bit next indicating whether to move left or right; namely $\delta(b, \mathsf{state}) = (\mathsf{state}', b', \mathsf{next})$. Let the movement function next be such that $\mathsf{next}(i)$ indicates whether the head on the input tape of M_O should move left or right after step i.

[9] It is not always necessary that we need to use the public key for FHE evaluation. Sometimes, a separate evaluation key for FHE evaluation alone is also used.

- Based on (M_O, next), construct a new Turing machine M_{FHE} that takes as input a FHE public key $\mathsf{PK}_{\mathsf{FHE}}$ and an input encryption $\widehat{x}$. M_{FHE} applies the transition function δ_{FHE} (the FHE evaluation of δ using $\mathsf{PK}_{\mathsf{FHE}}$) t times. Each cell of the tapes of M_O corresponds to an FHE encrypted value for M_{FHE}. The state of M_{FHE} at time i is the FHE encryption of the state of M_O corresponds to an FHE encrypted value for M_{FHE}. The state of M_{FHE} at time i is the FHE encryption of the state of M_O at time i. At step i, the transition function δ_{FHE} takes as input the encrypted bit from the input tape $\widehat{b}$ that the head currently points at, the current encrypted state $\widehat{\mathsf{state}}$ and outputs an encrypted new state $\widehat{\mathsf{state}'}$ and a new content $\widehat{b'}$. To determine whether to move the head left of right, compute $\mathsf{next}(i)$.
- Output the description of M_{FHE}.

The running time of $\mathsf{Compile}^{\mathsf{TM}}_{\mathsf{FHE}}$ and M_{FHE} is polynomial in t. The Turing machine M_{FHE} takes as input a public key and a ciphertext and then performs the FHE evaluation of M on the ciphertext. The resulting answer is then output by M_{FHE}.

B.3 Succinct Non Interactive Arguments of Knowledge

We now give background for succinct non-interactive arguments of knowledge. We present the details verbatim from Goldwasser et al. [GKP+13a]. We define the universal relation as a canonical form to represent verification-of-computation problems.

Definition 9. *[BCCT13] The universal relation is the set R_U of instance-witness pairs $(y, w) = ((U, x, t), w)$, where $|y|, |w| \leq t$ and U is a Turing machine, such that U accepts (x, w) after at most t steps. We denote by L_U the universal language corresponding to R_U. For any $c \in \mathbb{N}$, the universal NP relation is the set $R_{U,c}$, defined as R_U with the additional constraint that $t \leq |x|^c$.*

A SNARK is a triple of algorithms (Setup, P, V) that works as follows.

- The generator Setup on input the security parameter λ, samples a reference string CRS (since we consider publicly verifiable SNARKs, the CRS can also contain the public verification state). The Setup also takes as input a time bound B but we set this to $B = \lambda^{\log \lambda}$ which will never be achieved for NP language. Therefore, for simplicity, we do not make B explicit from now on.
- The honest prover $P(\mathsf{CRS}, y, w)$ produces a proof π for the statement $y = (U, x, t)$ given a valid witness w.
- The verifier $V(\mathsf{CRS}, y, \pi)$ takes as input the CRS, the instance y and a proof π and deterministically verifies π.

The SNARK is adaptive if the prover may choose the statement after seeing CRS.

Definition 10. *A triple of algorithms (Setup, P, V) for the relation $R_{U,c}$, where Setup is probabilistic and V is deterministic, is a SNARK if the following conditions are satisfied:*

- ***Completeness**: For every large enough security parameter $\lambda \in \mathbb{N}$, and for every instance-witness pair $(y, w) = ((U, x, t), w) \in R_U$,*

$$\mathsf{Prob}[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda);\ \pi \leftarrow P(\mathsf{CRS}, y, w)\ :\ V(\mathsf{CRS}, y, \pi)) = 1]$$

- ***Proof of knowledge**: For every polynomial-size prover P^* there exists a polynomial-size extractor Ext such that for every large enough security parameter $\lambda \in \mathbb{N}$, every auxillary input $z \in \{0,1\}^{poly(\lambda)}$, and every constant c:*

$$\mathsf{Prob}[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda); (y, \pi) \leftarrow P^*(\mathsf{CRS}, z);\ w \leftarrow \mathsf{Ext}(\mathsf{CRS}, z)\ :$$

$$V(\mathsf{CRS}, y, \pi) = 1 \text{ and } (y, w) \notin R_{U,c}] \ =\ negl(\lambda).$$

- ***Efficiency***. *There exists a universal polynomial p such that, for every large security parameter λ and every instance $y = (M, x, t)$,*
  1. *The generator* $\mathsf{Setup}(1^\lambda)$ *runs in time* $p(\lambda)$*;*
  2. *The prover* $P(\mathsf{CRS}, y, w)$ *runs in time* $p(\lambda, |U|, |x|, t)$*;*
  3. *The verifier* $V(\mathsf{CRS}, y, \pi)$ *runs in time* $p(\lambda, |U|, |x|)$*;*
  4. *An honestly generated proof has size* $p(\lambda)$*.*

Bitansky et al. [BCCT13] demonstrated a SNARK proof system under knowledge of exponent assumptions.

B.4 IND-CPA secure PKE

An IND-CPA (or semantically) secure Public Key Encryption scheme consists of three PPT algorithms $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ described as follows.

1. $\mathsf{KeyGen}(1^\lambda)$: On input 1^λ, it outputs public key $\mathsf{PK}_{\mathsf{PKE}}$ and decryption key $\mathsf{SK}_{\mathsf{PKE}}$.
2. $\mathsf{Encrypt}(m, \mathsf{PK}_{\mathsf{PKE}})$: On input message m and the public key, it outputs a ciphertext CT.
3. $\mathsf{Decrypt}(\mathsf{CT}, \mathsf{SK}_{\mathsf{PKE}})$: On input a ciphertext CT and the decryption key, it outputs m.

The IND-CPA scheme is said to be semantically secure if for any PPT adversary $\mathcal{A}$, there exists a negligible function α such that the following is satisfied for any two messages m_0, m_1 and for $b \in \{0, 1\}$:

$$|\mathsf{Prob}[\mathcal{A}(1^\lambda, \mathsf{Encrypt}(m_0, \mathsf{PK}_{\mathsf{PKE}})) = b] - \mathsf{Prob}[\mathcal{A}(1^\lambda, \mathsf{Encrypt}(m_1, \mathsf{PK}_{\mathsf{PKE}})) = b]| \leq \alpha(\lambda)$$

B.5 Simulation sound NIZK

We define the notion of simulation sound non-interactive zero knowledge [Sah99], which is a specific type of NIZK [BFM88] proof system. Intuitively, this notion says that there does not exist any efficient adversary even after receiving "fake" proofs for statements of his choice he cannot output any convincing proof, including fake proofs, for a statement for which he had not received any proof before.

More formally, a simulation-sound NIZK satisfies the following property along from the completeness and zero knowledge properties of any NIZK proof system. Consider the following game defined for PPT any adversary $\mathcal{A}$. The game begins with the execution of the simulator of the NIZK proof system who generates a fake CRS and a corresponding trapdoor. Then, $\mathcal{A}$ is given oracle access to a simulator which has a corresponding trapdoor. The adversary can submit any statement to this oracle and he will correspondingly get back a convincing proof (which is accepted by the NIZK verifier). The game ends with $\mathcal{A}$ outputting (x, Π). The adversary wins the game if (1) x was not equal to any of the statements he had queried the oracle and (2) x is not in the language for which the NIZK is defined and (3) Π is accepted by the NIZK verifier. We now define the simulation soundness property of a NIZK proof system.

Definition 11. *A NIZK proof system is said to satisfy simulation soundness if $\mathcal{A}$ wins the above game with negligible probability.*

C Proof of Security of $\mathsf{di}\mathcal{O}$ for TMs

We prove the correctness as well as the security of the $\mathsf{di}\mathcal{O}$ for TMs scheme presented in Section 3. Crucial to both these properties is the following lemma which shows that the program class can be implemented by a differing-inputs circuit family. To do this, we first define a PPT algorithm $\mathsf{Sampler}_{\mathcal{P}}^{\mathcal{M}}$ corresponding to the program class $\mathcal{P}$ as follows. The sampler $\mathsf{Sampler}_{\mathcal{P}}^{\mathcal{M}}$ receives

as input security parameter λ along with $(M_0, M_1, \mathsf{aux}_{\mathcal{M}})$, where $(M_0, M_1, \mathsf{aux}_{\mathcal{M}})$ is the output of $\mathsf{Sampler}_{\mathcal{M}}$, which is the sampler algorithm of $\mathcal{M}$. The sampler $\mathsf{Sampler}_{\mathcal{P}}^{\mathcal{M}}$ first executes the setup algorithm of FHE twice to obtain $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{SK}^1_{\mathsf{FHE}}), (\mathsf{PK}^2_{\mathsf{FHE}}, \mathsf{SK}^2_{\mathsf{FHE}})$. Then, the Turing machines M_0 and M_1 are encrypted using the public keys $\mathsf{PK}^1_{\mathsf{FHE}}$ and $\mathsf{PK}^2_{\mathsf{FHE}}$ to obtain g_1 and g_2 respectively. Finally, execute the setup algorithm of the SNARK proof system to obtain CRS. Output the programs $\mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^1_{\mathsf{FHE}},\mathsf{PK}^1_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})}$ and $\mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^2_{\mathsf{FHE}},\mathsf{PK}^1_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})}$. The auxillary information, denoted by $\mathsf{aux}_{\mathcal{P}}^{\mathcal{M}}$, consists of ($\mathsf{aux}_{\mathcal{M}}$, $\mathsf{PK}^1_{\mathsf{FHE}}$, $\mathsf{PK}^2_{\mathsf{FHE}}$, CRS).

Lemma 1. *Consider a class of programs $\mathcal{P}$, defined as before. Let $\mathsf{P1} = \mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^1_{\mathsf{FHE}},\mathsf{PK}^1_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})}$ and $\mathsf{P2} = \mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^2_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})}$ along with auxillary information $\mathsf{aux}_{\mathcal{P}}^{\mathcal{M}}$ be the output of sampler algorithm $\mathsf{Sampler}_{\mathcal{P}}^{\mathcal{M}}$. There does not exist any PPT adversary $\mathcal{A}$ on input $(\mathsf{P1}, \mathsf{P2}, \mathsf{aux}_{\mathcal{P}}^{\mathcal{M}})$ outputs y such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$, with non-negligible probability under the assumption that $\mathcal{M}$ is a differing-inputs Turing machine family.*

To prove this, we assume that there exists an adversary $\mathcal{A}$ that outputs y such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$. Using $\mathcal{A}$, we construct another adversary $\mathcal{A}_{\mathcal{M}}$ which violates the differing-inputs property of $\mathcal{M}$ arriving at a contradiction. Before we proceed further, we make a notational simplification. We assume that the input to $\mathsf{P1}$ (or $\mathsf{P2}$) can be parsed as (z, φ), where $z = (i, e_1^{(i)}, e_2^{(i)}, g_1, g_2, h_x, t, \varphi)$.

We present the following claims that will be useful when we calculate the probability of success of $\mathcal{A}_{\mathcal{M}}$. The first claim says that if the programs $\mathsf{P1}$ and $\mathsf{P2}$ differ on any input $y = (z, \varphi)$ then the verifiers both in $\mathsf{P1}$ and $\mathsf{P2}$ accept the proof φ. Recall that any program in $\mathcal{P}$ has two components, namely the SNARK verifier along with FHE decryption circuit. The second claim states that if both the programs differ on any input (z, φ) then the output of $\mathsf{P1}$ (resp., $\mathsf{P2}$) is the decryption of $e_1^{(i)}$ (resp., $e_2^{(i)}$). Hence, as a consequence, we have that the decryption of $e_1^{(i)}$ (with respect to $\mathsf{PK}^1_{\mathsf{FHE}}$) different from the decryption of $e_2^{(i)}$ (with respect to $\mathsf{PK}^2_{\mathsf{FHE}}$). We now state the claims.

Claim 11. If there exists a $y = (z, \varphi)$ such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$ then the verifier in both $\mathsf{P1}$ and $\mathsf{P2}$ accept φ.
Proof. The first observation is that the verifier as part of $\mathsf{P1}$ is same as the verifer which is part of $\mathsf{P2}$. The second observation is that the verifier in $\mathsf{P1}$ (resp., $\mathsf{P2}$) does not reject the proof φ. This is because, if the verifier in $\mathsf{P1}$ (resp., $\mathsf{P2}$) rejects then the output of $\mathsf{P1}$ (resp., $\mathsf{P2}$) is 0 which will contradict our hypothesis that the output of the two programs are different.

Claim 12. If there exists $y = (i, e_1^{(i)}, e_2^{(i)}, h_x, t, \varphi)$ such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$ then the output of $\mathsf{P1}$ (resp., $\mathsf{P2}$) is the decryption of $e_1^{(i)}$ (resp., $e_2^{(i)}$) with respect to $\mathsf{PK}^1_{\mathsf{FHE}}$ (resp., $\mathsf{PK}^2_{\mathsf{FHE}}$).
Proof. From Claim 11, we have that the verifiers as part of both $\mathsf{P1}$ and $\mathsf{P2}$ accept and hence, the output of $\mathsf{P1}(y)$ is the decryption of $e_1^{(i)}$ with respect to $\mathsf{PK}^1_{\mathsf{FHE}}$ and similarly, the output of $\mathsf{P2}$ is the decryption of $e_2^{(i)}$ with respect to $\mathsf{PK}^2_{\mathsf{FHE}}$.

Now, consider the following adversary. Since (P, V) which is a SNARK system has knowledge extractability property we assume that there exists an extractor $\mathsf{Ext} = (\mathsf{Ext}_1, \mathsf{Ext}_2)$ such that Ext_1 generates $(\mathsf{CRS}, \mathsf{state})$ and then Ext_2 on input an instance, $(\mathsf{CRS}, \mathsf{state})$ along with a proof, extracts a witness corresponding to that instance [10].

[10]We emphasise that this is the only place where we need the extractability property. As mentioned a couple of times before, if the length of the witness (which in this case is the input to M_0 or M_1) is bounded above apriori then we could have just used SNARGs for our construction

$\underline{\mathcal{A}_{\mathcal{M}}(M_0, M_1, \mathsf{aux}_{\mathcal{M}})}$:
$(\mathsf{SK}^1_{\mathsf{FHE}}, \mathsf{PK}^1_{\mathsf{FHE}}) \leftarrow \mathsf{Setup}_{\mathsf{FHE}}(1^\lambda)$
$(\mathsf{SK}^2_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}) \leftarrow \mathsf{Setup}_{\mathsf{FHE}}(1^\lambda)$
$(\mathsf{CRS}, \mathsf{td}) \leftarrow \mathsf{Ext}_1(1^\lambda)$
$g_1 \leftarrow \mathsf{Encrypt}_{\mathsf{FHE}}(\mathsf{PK}^1_{\mathsf{FHE}}, M_0)$
$g_2 \leftarrow \mathsf{Encrypt}_{\mathsf{FHE}}(\mathsf{PK}^2_{\mathsf{FHE}}, M_1)$
Denote $\mathsf{di}\mathcal{O}(\lambda, \mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^1_{\mathsf{FHE}},\mathsf{PK}^1_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})})$ by $\mathsf{P1}$
Denote $\mathsf{di}\mathcal{O}(\lambda, \mathsf{P}^{(g_1,g_2,\mathsf{CRS})}_{(\mathsf{SK}^2_{\mathsf{FHE}},\mathsf{PK}^1_{\mathsf{FHE}},\mathsf{PK}^2_{\mathsf{FHE}})})$ by $\mathsf{P2}$
$y \leftarrow \mathcal{A}(\mathsf{P1}, \mathsf{P2}, \mathsf{aux}^{\mathcal{M}}_{\mathcal{P}} = (\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2, \mathsf{CRS}))$
Parse y as (z, φ)
$x \leftarrow \mathsf{Ext}_2(z, \varphi, \mathsf{CRS}, \mathsf{td})$
Output x

The next claim shows that if $\mathcal{A}$ can produce an input y such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$ with non-negligible probability then $\mathcal{A}_{\mathcal{M}}$ violates the security game of the differing-inputs corresponding to the family $\mathcal{M}$ (Definition 4). Before we go ahead and prove the claim, we first observe that the distribution of $(\mathsf{P1}, \mathsf{P2}, \mathsf{aux}^{\mathcal{M}}_{\mathcal{P}})$ as input to $\mathcal{A}$ in the description of $\mathcal{A}_{\mathcal{M}}$ is the same as the output distribution of $\mathsf{Sampler}^{\mathcal{M}}_{\mathcal{P}}$.

Claim 13. Let $(\mathsf{P1}, \mathsf{P2}, \mathsf{aux}^{\mathcal{M}}_{\mathcal{P}}) \leftarrow \mathsf{Sampler}^{\mathcal{M}}_{\mathcal{P}}(1^\lambda)$. If there exists an adversary $\mathcal{A}$ on input $(\mathsf{P1}, \mathsf{P2}, \mathsf{aux})$ outputs y with non-negligible probability then the adversary $\mathcal{A}_{\mathcal{M}}$ on input (M_0, M_1), which is the output of $\mathsf{Sampler}_{\mathcal{M}}(1^\lambda)$, produces x such that $M_0(x) \neq M_1(x)$.
Proof. Suppose the adversary $\mathcal{A}$ outputs y such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$ with non-negligible probability. We make the following two observations that will prove the above claim.

- Using Claim 11, we have the fact that the verifier accepts the proof φ corresponding to the instance z with non-negligible probability, where y can be parsed as (z, φ) is the output of $\mathcal{A}$. From the knowledge extractability property, we have the fact that the extractor outputs a valid witness x with non-negligible probability. Since, x is a valid witness to z, we have the fact that $e^{(i)}_1$ is an encryption of $M_0(x)$ with respect to public key $\mathsf{PK}^1_{\mathsf{FHE}}$ and similarly, $e^{(i)}_2$ is an encryption of $M_1(x)$ with respect to public key $\mathsf{PK}^2_{\mathsf{FHE}}$.
- Using Claim 12, we have that the output of program $\mathsf{P1}$ (resp., $\mathsf{P2}$) is the decryption of g_1 (resp., g_2) with respect to $\mathsf{PK}^1_{\mathsf{FHE}}$ (resp., $\mathsf{PK}^2_{\mathsf{FHE}}$). Rephrasing this in terms of first observation, the output of $\mathsf{P1}$ on input y is $M_0(x)$ and the output of $\mathsf{P2}$ on input y is $M_1(x)$. Since, y is such that $\mathsf{P1}(y) \neq \mathsf{P2}(y)$ with non-negligible probability, we have that $M_0(x) \neq M_1(x)$. This completes the proof.

The above claim contradicts the assumption that $\mathcal{M}$ is a differing-inputs Turing machine family and this proves $\mathcal{P}$ is a differing-inputs circuit family.

Corollary 2. *Differing-inputs obfuscation for the circuit family $\mathcal{P}$ exists under the assumption that IND-CPA FHE exists, SNARKs exists, collision resistant hash functions and differing-inputs obfuscation exists for any differing-inputs circuit family.*

We now prove the correctness and the security of the differing-inputs obfuscation scheme.

Correctness. For simplicity, we assume that the obfuscaton of $\mathsf{P1}$, denoted by $\mathsf{P1}_{\mathsf{obf}}$, is executed once, as against multiple times, and the entire output of the FHE evaluation phase is fed to the obfuscation of $\mathsf{P1}$. We now argue the correctness of the scheme. The correctness of the FHE scheme along with the correctness of the SNARK proof system imply that the input to $\mathsf{P1}$ is an encryption of $M(x)$ followed by a valid proof that the encryption is correctly computed, where x is the input to the obfuscation scheme. Now, note that if a valid encryption of $M(x)$

and a valid proof that $M(x)$ is correctly computed is given to $\mathsf{P1}$ then the output of $\mathsf{P1}$ would be $M(x)$. And so, by the correctness of $\mathsf{di}\mathcal{O}$ it follows that even the output of the obfuscation of $\mathsf{P1}$, which is $\mathsf{P1}_{\mathsf{obf}}$, is $M(x)$. This proves the correctness of the $\mathsf{di}\mathcal{O}$ scheme.

Security proof. We now describe the security proof of the differing-inputs obfuscation scheme for the Turing machines. The security expermient proceeds by the challenger first executing the sampler algorithm of $\mathcal{M}$. On input a security parameter, the sampler algorithm outputs $(M_0, M_1, \mathsf{aux}_{\mathcal{M}})$. The challenger then sends M_{obf} to the adversary, where M_{obf} is either the $\mathsf{di}\mathcal{O}$ obfuscation of M_0 or M_1. The security guarantee is that the adversary's output when M_0 is obfuscated is negligibly close to its output when M_1 is obfuscated. To show this, we first describe the hybrids which are similar to the hybrids in the security proof of the indistinguishability obfuscation scheme of the circuits from Garg et al. [GGH+13b]. For completeness sake, we present the hybrids below.

Hybrid_0: This corresponds to the honest execution of the differing-inputs obfuscation corresponding to the Turing machine M_0.

Hybrid_1: In this hybrid, the ciphertext g_1 is generated by encrypting M_0 (under $\mathsf{PK}^1_{\mathsf{FHE}}$) while the ciphertext g_2 is obtained by encrypting the Turing machine M_1 (under $\mathsf{PK}^2_{\mathsf{FHE}}$). The rest of the hybrid is the same as the previous hybrid Hybrid_0.

Hybrid_2: The ciphertexts g_1 and g_2 are generated the same way as in the previous hybrid. The only difference is that instead of obfuscating program $\mathsf{P1}$, the program $\mathsf{P2}$ is obfuscated.

Hybrid_3: In this hybrid, the ciphertexts g_1 is generated by encrypting M_1 (under $\mathsf{PK}^1_{\mathsf{FHE}}$) while the ciphertext g_2 is (still) generated by encrypting M_1 (under $\mathsf{PK}^2_{\mathsf{FHE}}$). As in the previous hybrid, the obfuscation component is still generated from $\mathsf{P2}$.

Hybrid_4: The ciphertexts are generated as in the previous hybrid. That is, g_1 and g_2 are encryptions of M_1 under keys $\mathsf{PK}^1_{\mathsf{FHE}}$ and $\mathsf{PK}^2_{\mathsf{FHE}}$ respectively. But this time, the obfuscation component corresponds to the program $\mathsf{P1}$ instead of $\mathsf{P2}$.

Note that this corresponds to the honest execution of the differing-inputs obfuscation corresponding to the obfuscation of M_1.

We present a series of claims that show that the hybrids are computationally indistinguishable with respect to each other.

Claim 1. Hybrids H_0 and H_1 are computationally indistinguishable under the assumption that the FHE scheme is IND-CPA secure.

Proof. We assume that these two hybrids are distinguishable and then arrive at a contradiction by contradicting the IND-CPA security of the FHE scheme. Suppose there exists an adversary $\mathcal{A}$ that distinguishes hybrids Hybrid_0 and Hybrid_1 then we construct an adversary $\mathcal{A}'$ that breaks the IND-CPA security of FHE scheme as follows. The adversary $\mathcal{A}'$, on input a public key PK_1, first executes $\mathcal{A}$ to get the messages M_0 and M_1. It then sends this to the challenger who decides to encrypt either M_0 or M_1 depending on the challenge bit. The challenege ciphertext, $\mathsf{CT}^{(2)}$, is handed over to $\mathcal{A}'$ who does the following. It generates public key-secret key pair $(\mathsf{SK}_0, \mathsf{PK}_0)$ and then encrypts m_0 using PK_0 to obtain $\mathsf{CT}^{(1)}$. Further, it generates the program $\mathsf{P1}$ in which the decryption is done using the decryption key SK_0. It finally gives $(\mathsf{CT}^{(1)}, \mathsf{CT}^{(2)}, \mathcal{P}1_{\mathsf{obf}})$, where $\mathsf{P1}_{\mathsf{obf}} = \mathsf{di}\mathcal{O}(\mathsf{P1})$, to $\mathcal{A}$ and then $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

Claim 2. Hybrids H_1 and H_2 are computationally indistinguishable under the assumption that

differing-inputs obfuscators exist for all circuits.
Proof. Consider an adversary who receives $\mathsf{P1}, \mathsf{P2}$ and $\mathsf{P}_{\mathsf{obf}}$, which is either an obfuscation of $\mathsf{P1}$ or $\mathsf{P2}$. If the adversary receives an obfuscation of $\mathsf{P1}$ then we are in hybrid Hybrid_1 and if the adversary receives the obfuscation of $\mathsf{P2}$ then we are in hybrid Hybrid_2. So, if the adversary could indeed distinguish the two hybrids with non-negligible probability then he can as well distinguish the obfuscations of $\mathsf{P1}$ and $\mathsf{P2}$ with non-negligible probability. This contradicts the differing-inputs property of $\mathcal{P}$ from Corollary 2, thus proving the claim.

Claim 3. Hybrids H_2 and H_3 are computationally indistinguishable under the assumption that the FHE scheme is IND-CPA secure.
Proof. We assume that these two hybrids are distinguishable and then arrive at a contradiction by contradicting the IND-CPA security of the FHE scheme. Suppose there exists an adversary $\mathcal{A}$ that distinguishes hybrids Hybrid_0 and Hybrid_1 then we construct an adversary $\mathcal{A}'$ that breaks the IND-CPA security of FHE scheme as follows. The adversary $\mathcal{A}'$, on input a public key PK_0, first executes $\mathcal{A}$ to get the messages m_0 and m_1. It then sends this to the challenger who decides to encrypt either m_0 or m_1 depending on the challenge bit. The challenge ciphertext, $\mathsf{CT}^{(1)}$, is handed over to $\mathcal{A}'$ who does the following. It generates public key-secret key pair $(\mathsf{SK}_1, \mathsf{PK}_1)$ and then encrypts m_1 using PK_1 to obtain $\mathsf{CT}^{(2)}$. Further, it generates the program $\mathsf{P2}$ in which the decryption is done using the decryption key SK_2. Finally, it computes $\mathcal{P}2_{\mathsf{obf}}$, which is the indistinguishability obfuscation of $\mathsf{P2}$. It gives $(\mathsf{CT}^{(1)}, \mathsf{CT}^{(2)}, \mathsf{P2}_{\mathsf{obf}})$ to $\mathcal{A}$ and then $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

Claim 4. Hybrids H_3 and H_4 are computationally indistinguishable under the assumption that differing-inputs obfuscators exist for all circuits.
Proof. This is similar to the proof of Claim 2. Consider an adversary who receives $\mathsf{P1}, \mathsf{P2}$ and $\mathsf{P}_{\mathsf{obf}}$, which is either an obfuscation of $\mathsf{P1}$ or $\mathsf{P2}$. If the adversary receives an obfuscation of $\mathsf{P2}$ then we are in hybrid Hybrid_3 and if the adversary receives the obfuscation of $\mathsf{P1}$ then we are in hybrid Hybrid_4. So, if the adversary could indeed distinguish the two hybrids with non-negligible probability then he can as well distinguish the obfuscations of $\mathsf{P1}$ and $\mathsf{P2}$ with non-negligible probability. This contradicts the differing-inputs property of $\mathcal{P}$ from Corollary 2, thus proving the claim.

From the above arguments it follows that hybrids Hybrid_0 and Hybrid_4 are computationally indistinguishable. This proves the differing-inputs property of the Turing machines M_0 and M_1. More formally,

Theorem 3. *Under the existence of the following primitives, the construction in Section 3 is a differing-inputs obfuscation for any family of differing-inputs Turing machines.*

- *IND-CPA secure fully homomorphic encryption scheme.*
- *Succinct Non-Interactive Arguments of Knowledge.*
- *Differing-inputs obfuscation for all circuits.*
- *Collision resilient hash functions.*

D FE for Turing machines

In this section, we give a construction of a functional encryption scheme that enjoys the succinctness as well as the input-specific runtime properties. The scheme uses indistinguishability obfuscation for Turing machines as a building block. Note that our construction in Section 3 applies for indistinguishability obfuscation for Turing machines as well. We adopt the scheme as described in Garg et al. [GGH+13b] to the case of Turing machines. Similar to Garg et al. we are

only able to achieve selective security. However using complexity leveraging we can boost this to full security [11]. We present the scheme as presented in [GGH+13b] below. The construction uses public key encryption, statistically simulation sound non-interactive zero knowledge proofs. The background for these primitives can be found in Garg et al. We let (Setup_{PKE}, $\mathsf{Encrypt}_{PKE}$, $\mathsf{Decrypt}_{PKE}$) be the algorithms comprising our (perfectly correct) encryption scheme. Our SSS-NIZK system will consist of algorithms Setup_{NIZK}, Prove_{NIZK}, Verify_{NIZK} and has a simulator Sim.

We build a functional encryption system for messages of length $n = n(\lambda)$. For messages of length n and security parameter λ the ciphertexts of our PKE scheme will be of length $\ell = \ell(\lambda, n)$. The construction is as follows:

- $\mathsf{Setup}_{\mathsf{FE}}(1^\lambda)$: The $\mathsf{Setup}_{\mathsf{FE}}$ algorithm takes the security parameter λ and computes the following.
  1. Generate $(\mathsf{PK}^1_{PKE}, \mathsf{SK}^1_{PKE}) \leftarrow \mathsf{Setup}_{PKE}(1^\lambda)$ and $(\mathsf{PK}^2_{PKE}, \mathsf{SK}^2_{PKE}) \leftarrow \mathsf{Setup}_{PKE}(1^\lambda)$.
  2. Set $\mathsf{CRS} \leftarrow \mathsf{Setup}_{NIZK}$.

  It sets the public parameters and master secret key as

$$\mathsf{PP} = \{\mathsf{PK}^1_{PKE}, \mathsf{PK}^2_{PKE}, \mathsf{CRS}\} \quad \text{and} \quad \mathsf{MSK} = \{\mathsf{SK}^1_{PKE}\}.$$

- $\mathsf{KeyGen}_{\mathsf{FE}}(\mathsf{MSK}, f)$: On input the master secret key and a Turing machine implementing f, it does the following. It computes an indistinguishability obfuscation (for Turing machines) of $\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathsf{CRS})}$, denoted by $\mathsf{O}_{\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathsf{CRS})}}$, for the program $\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathsf{CRS})}$ using the size of the Turing machine implementing $\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathsf{CRS})}$ to be equal to the value $\max\{|\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathsf{CRS})}|, |\mathrm{P4}^{(f,\mathsf{SK}^2_{PKE},\mathsf{CRS})}|\}$. We output $\mathsf{O}_{\mathrm{P3}^{(f,\mathsf{SK}^1_{PKE},\mathsf{CRS})}}$ as the secret key SK_f.
- $\mathsf{Encrypt}_{\mathsf{FE}}(\mathsf{PP}, m \in \{0,1\}^n)$: On input the public parameters PP and a message m, output $c := (e_1, e_2, \pi)$, where $e_1 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1)$ and $e_2 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)$ and π is a NIZK proof of Equation 2.
- $\mathsf{Decrypt}_{\mathsf{FE}}(\mathsf{SK}_f, c = (e_1, e_2, \pi))$: The decryption algorithm runs the obfuscated Turing machine SK_f on input (e_1, e_2, π) and outputs the answer.

The two program classes as mentioned in the construction are described below.

[11] We note that the construction of the adaptively-secure FE scheme as described in [BCP14] does not use complexity leveraging.

P3

On input(e_1, e_2, π), the program P3$^{(f, \mathsf{SK}^1_{PKE}, \mathsf{CRS})}$ proceeds as follows:

1. Check that π is valid NIZK proof (using the Verify_{NIZK} algorithm and CRS) for the NP-statement

$$\exists m, r_1, r_2 : \qquad (2)$$
$$\Big(e_1 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1) \bigwedge e_2 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)\Big)$$

2. If any checks fail output 0; otherwise output $f\big(\mathsf{Decrypt}_{PKE}(\mathsf{SK}^1_{PKE}, e_1)\big)$.

P4

On input(e_1, e_2, π), P4$^{(f, \mathsf{SK}^2_{PKE}, \mathsf{CRS})}$, the program proceeds as follows:

1. Check that π is valid NIZK proof (using the Verify_{NIZK} algorithm and CRS) for the NP-statement

$$\exists m, r_1, r_2 :$$
$$\Big(e_1 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^1_{PKE}, m; r_1) \bigwedge e_2 = \mathsf{Encrypt}_{PKE}(\mathsf{PK}^2_{PKE}, m; r_2)\Big)$$

2. If any checks fail output 0; otherwise output $f\big(\mathsf{Decrypt}_{PKE}(\mathsf{SK}^2_{PKE}, e_2)\big)$.

We do not describe the correctness and the security arguments here since they are identical to the arguments in Garg et al. [GGH+13b]. Since the functional keys are (indistinguishability) obfuscation of Turing machines, it can be seen that the scheme satisfies both succinctness and input-specific runtime properties.

E Delegatable functional encryption scheme

The notion of delegatable functional encryption scheme is introduced in this section. Delegatable functional encryption is a functional encryption scheme having the additional operation of delegation of functional keys. We first give an informal description of the delegate operation. Suppose, Alice has a functional key corresponding to some function. Alice decrypting all the messages all by herself is cumbersome. She wants to delegate some specific decryptions to Bob. One way to do that is Alice hands over her key to Bob. However, Bob can now decrypt messages which he is not supposed to. Instead what Alice can do is compute a new key from the functional key it possesses and she can hand over the key to Bob. The key is designed in such a way that Bob can only decrypt messages he is supposed to and nothing more. This is precisely what delegation deals with. We define the class of functions that can be delegated. Suppose, Alice has a key corresponding to function f then she can delegate those functions g which can be written as a composition of f' on f, denoted by $f' \circ f$, for some function f' [12].

[12] More formally, $f' \circ f$ takes as input x and outputs $f'(f(x))$.

E.1 Definition

We define a delegatable functional encryption scheme to consist of the following PPT algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Delegate}$). The first four PPT algorithms are the same as in the definition of the functional encryption described in Appendix A.2.

- $\mathsf{Setup}(1^\lambda)$ - a polynomial time algorithm that takes the unitary representation of the security paramter λ and outputs a public paramteres PP and a master secret key MSK.
- $\mathsf{KeyGen}(\mathsf{MSK}, f)$ - a polynomial time algorithm that takes as input the master secret key MSK and a function f implementable by a Turing machine $M \in \mathcal{M}$ and outputs a corresponding secret key SK_f.
- $\mathsf{Encrypt}(\mathsf{PP}, x)$ - a polynomial time algorithm that takes the public parameters PP and a string $x \in \mathcal{S}$ and outputs a ciphertext CT.
- $\mathsf{Decrypt}(\mathsf{SK}_f, \mathsf{CT})$ - a polynomial time algorithm that takes a secret key SK_f and ciphertext encrypting message $x \in \mathcal{S}$ and outputs $f(x)$.
- $\mathsf{Delegate}(\mathsf{PP}, \mathsf{SK}_f, f')$ - a polynomial time algorithm that takes as input a public key PP, a functional key SK_f and a function f' and outputs a functional key $\mathsf{SK}_{f' \circ f}$ that evaluates the function $f' \circ f$ on the message contained in the ciphertext.

A delegatable functional encryption scheme satisfies two main properties, namely correctness and security. The criterion for correctness is the same as that of the functional encryption scheme. In addition, the following must be satisfied – if the output of a delegate operation on input SK_f and f' is $\mathsf{SK}_{f' \circ f}$ then the decryption algorithm on input $\mathsf{SK}_{f' \circ f}$ along with an encryption of a message x should give $f'(f(x))$ as its output. We describe the security notion next.

E.2 Security notion

We now describe the security notion employed for a delegatable encryption scheme. We follow the security notion defined in [SW08] for a predicate encryption scheme. The security is modelled as a game between a challenger and an adversary.

Setup. The challenger executes the Setup algorithm of the delegatable functional encryption scheme and gives the public key, denoted by PK to the adversary.

Query. The adversary submits queries to the challenger adaptively. There are three subphases in the query phase. The first is the creation of the functional key, second is the delegation phase and the third is the reveal phase.

- *Creation.* The adversary submits the queries f_i to the challenger who computes the keys SK_{f_i} corresponding to f_i. These keys are not yet revealed to the adversary.
- *Delegation.* The adversary now chooses the keys, generated during the creation phase, on which the delegation operation need to be applied. This is done by the adversary submitting a function f_i' along with an index i, to indicate the key on which the delegate operation need to applied. The challenger then executes $\mathsf{Delegate}$ on SK_{f_i} along with f_i' to obtain $\mathsf{SK}_{f_i' \circ f_i}$. As in the previous case, the key is not yet revealed to the adversary.
- *Reveal.* The adversary asks the challenger to reveal a functional key that was generated in one of the previous phases.

Challenge. The adversary then sends the two challenge messages x_0, x_1 such that for all functional keys SK_f created by the challenger (including the ones during the delegation phase), $f(x_0) = f(x_1)$. If this condition is not satisfied then the challenger aborts the game. Otherwise,

the challenger encrypts x_b using the public key PK, where b is a bit chosen uniformly at random, and the resulting ciphertext is then handed over to the adversary.

Query. This phase is similar to the previous query phase. In this phase too, for any functional key SK_f created, $f(x_0)$ should be the same as $f(x_1)$.

Guess. The game ends when the adversary guesses a bit b'. The advantage of an adversary in the above game is defined to be $|\mathsf{Prob}[b' = b] - \frac{1}{2}|$.

Definition 12. *A delegatable functional encryption scheme is said to be (fully) secure if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is a negligible function of λ.*

We consider a weaker notion of security, namely selective security, where the challenge messages x_0 and x_1 are fixed by the adversary before the Setup phase is executed. We note that we can transform a selectively secure delegatable FE scheme to an adaptively secure delectable FE scheme using complexity leveraging.

E.3 Construction

Consider the functional encryption scheme described in Appendix D. Corresponding to this scheme we define a delegate operation, denoted by $\mathsf{Delegate}$, as follows. On input a key SK_f and a function f', specified by a Turing machine, the delegate operation computes the indistinguishability obfuscation of the program $\mathsf{P}^{f',\mathsf{SK}_f}$. The program $\mathsf{P}^{f',\mathsf{SK}_f}$ on input $(\mathsf{CT}_1, \mathsf{CT}_2, h_\Pi, \varphi)$, first evaluates the obfuscation SK_f on input $(\mathsf{CT}_1, \mathsf{CT}_2, h_\Pi)$ to obtain z. It then evaluates f' on z to obtain $f'(z)$, which it then outputs.

We claim that this is a delegatable encryption scheme. The correctness of this scheme follows from the correctness of indistinguishability obfuscation. We argue about the security informally here. We first design the hybrids such that the first hybrid corresponds to the indistinguishability game in the delegatable functional encryption scheme and the last hybrid corresponds to the game in the functional encryption scheme. In each hybrid, we replace a delegate operation by a key generation operation. That is, if an adversary requests delegate operation f' on key SK_f, instead of performing the delegation operation we generate a fresh key $\mathsf{SK}_{f' \circ f}$.

To argue the indistinguishability of the hybrids, note that it suffices to show that the output distribution of the key generation for the function $f' \circ f$ is computationally indistinguishable from the output distribution of the delegation operation on input SK_f, corresponding to f, and f'. To see this, observe that the programs $\mathsf{P}^{f',\mathsf{SK}_f}$ and $\mathsf{P}^{(f,\mathsf{SK}_1,\mathsf{CRS}_{\mathcal{SS}},\mathsf{CRS}_{\mathsf{SNARK}})}$ are equivalent. The output of the key generation is the obfuscation of $\mathsf{P}^{(f,\mathsf{SK}_1,\mathsf{CRS}_{\mathcal{SS}},\mathsf{CRS}_{\mathsf{SNARK}})}$ and correspondingly the output of the delegate operation is the output of $\mathsf{P}^{f',\mathsf{SK}_f}$. From the equivalence of these programs, it follows that their obfuscations, and hence the outputs (distributions) of KeyGen and $\mathsf{Delegate}$ are computationally indistinguishable. This proves that any two consecutive hybrids are computationally indistinguishable. The adversary can succeed in the last hybrid with only negligible probability and this follows from the fact that our functional encryption scheme is secure. Hence, the adversary can succeed in the first hybrid, which is the security game of the delegatable encryption scheme, only with negligible probability. This completes the proof.

Remark. The above delegatable functional encryption scheme works for both circuits as well as Turing machines. If a delegatable scheme need to be constructed only for circuits then we can directly construct the scheme from the functional encryption scheme by Garg et. al. [GGH+13b] using the delegation operation defined as above.

F Multiparty Key Exchange and Broadcast Encryption

Here we prove Theorem 1, which states that our multiparty key exchange protocol is secure. We also our broadcast encryption construction and prove its security.

We start with the proof of Theorem 1.

Proof. We prove security through a sequence of hybrids.
Hybrid_0: This is the honest key exchange game, $\texttt{EXP}(0)$ in the NIKE security definition, where the adversary receives an obfuscation of $\mathsf{P}^{(F)}$, published values $\{y_i\}_{i=1,\ldots,n}$, and the correct challenge group key $k^* = F(h^*)$ where $h^* = \mathcal{H}(y_1, \ldots, y_n)$.
Hybrid_1: This game is identical to Hybrid_0, except that instead of receiving the correct public key consisting of $\mathsf{P1}_{\mathsf{obf}}$, the adversary receives the obfuscation $\mathsf{P2}_{\mathsf{obf}}$ of the program $\mathsf{P}_2^{(h^*, F^{h^*})}$ in Figure 3.

$\mathsf{P}_2^{(h^*, F^{h^*})}$

Given input (h, π, i, y, x), $\mathsf{P}_2^{(h^*, F^{h^*})}$ proceeds as follows:

1. Check that $h \neq h^*$.
2. Check that π is a valid proof that there exists $z \in \mathcal{Y}^n$ where $\mathcal{H}(z) = h$ and $z[i] = y$.
3. Check that $f(x) = y$.
4. If any check fails, abort and output $\perp$.
5. Otherwise, output $F^{h^*}(h)$

Figure 3 The program $\mathsf{P}_2^{(h^*, F^{h^*})}$ that users will use for key generation.

Hybrid_2: This game is identical to Hybrid_1, except that instead of setting the challenge group key as $k^* = F(h^*)$, this k^* is chosen uniformly at random from the range of F, independent of F.
Hybrid_3: This game is identical to Hybrid_2, except the adversary is given the correct public key consisting of an obfuscation of $\mathsf{P}^{(F)}$. This game is the same as Hybrid_0, except that the challenge group key k^* is chosen uniformly at random, and is therefore identical to $\texttt{EXP}(1)$ in the NIKE security definition.

We need to argue that each of these hybrids are indistinguishable. First, we argue that $\mathsf{P}^{(F)}$ and $\mathsf{P}^{h^*, F^{h^*}}$ form a differing-inputs circuit family, where $h^* = \mathcal{H}(y_1, \ldots, y_n)$ where $y_i = f(x_i)$ and the adversary gets auxiliary information $\{y_i\}_{i=1,\ldots,n}$. Consider any differing input (h, π, i, y, x). It must be that $h = h^*$, π is a valid proof, and $f(x) = y$. There are two cases:

- $y = y_i$. Then x is a pre-image of y_i under f. We can use such a differing input to break the one-wayness of f.
- $y \neq y_i$. Then since π is valid, but $h^* = \mathcal{H}(y_1, \ldots, y_n)$ with $y_i \neq y$, the proof must yield a collision on the underlying collision resistant hash H.

Therefore, the security of f and $\mathcal{H}$ show that P and P_2 form a differing-inputs function family. Therefore, the obfuscations $\mathsf{P1}_{\mathsf{obf}}$ and $\mathsf{P2}_{\mathsf{obf}}$ are indistinguishable. This in turn shows that Hybrid_0 is indistinguishable from Hybrid_1. The same applies to Hybrid_2 and Hybrid_3. Therefore, it remains to prove the indistinguishability of Hybrid_1 and Hybrid_2.

Let $\mathcal{A}$ be an adversary that distinguishes Hybrid_1 from Hybrid_2 with probability ϵ. We construct an adversary $\mathcal{B}$ that breaks the security of F. $\mathcal{B}$ generates x_i for himself and computes $y_i = f(x_i)$. $\mathcal{B}$ also computes $h^* = \mathcal{H}(y_1, \ldots, y_n)$, and then asks its challenger for the constrained

function F^{h^*} and the value of F at h^*, obtaining the key k^*. Now $\mathcal{B}$ constructs the obfuscation of P_2. It gives this obfuscation, all of the y_i, and k^* to $\mathcal{A}$, and runs $\mathcal{A}$. $\mathcal{B}$ outputs the output of $\mathcal{A}$.

If k^* is the correct value of $F(h^*)$, then $\mathcal{B}$ correctly simulates Hybrid_1. Otherwise if k^* is random it simulates Hybrid_2. Therefore, $\mathcal{B}$ breaks the security of F with probability ϵ, meaning ϵ is negligible. Thus Hybrid_1 is indistinguishable from Hybrid_2, as desired.

We have thus shown that Hybrid_0 is indistinguishable from Hybrid_3, showing that our construction is statically secure. □

F.1 Broadcast Encryption

Next, we construct a broadcast encryption system with short ciphertexts, public keys, and secret keys. We begin by defining a broadcast encryption scheme and what it means to be secure. A (public-key) broadcast encryption system [FN94] is made up of three randomized algorithms:

$\mathsf{Setup}(\lambda, n)$ Given the security parameter λ and the number of receivers n, output n private keys $\mathsf{SK}_1, \ldots, \mathsf{SK}_n$ and public parameters PP. For $i = 1, \ldots, n$, recipient number i is given the private key SK_i.

$\mathsf{Encrypt}(\mathsf{PP}, S)$ Takes as input a subset $S \subseteq \{1, \ldots, n\}$, and the public parameters PP. It outputs a pair (Hdr, k) where Hdr is called the header and $k \in \mathcal{K}$ is a message encryption key chosen from a key space $\mathcal{K}$. We will often refer to Hdr as the broadcast ciphertext.

Let m be a message to be broadcast that should be decipherable precisely by the receivers in S. Let c_m be the encryption of m under the symmetric key k. The broadcast data consists of (S, Hdr, c_m). The pair (S, Hdr) is often called the full header and c_m is often called the broadcast body.

$\mathsf{Decrypt}(\mathsf{PP}, i, \mathsf{SK}_i, S, \mathsf{Hdr})$ Takes as input a subset $S \subseteq \{1, \ldots, n\}$, a user id $i \in \{1, \ldots, n\}$ and the private key SK_i for user i, and a header Hdr. If $i \in S$ the algorithm outputs a key $k \in \mathcal{K}$. Intuitively, user i can then use k to decrypt the broadcast body c_m and obtain the message m.

The above definition describes a public-key broadcast encryption scheme. In a secret-key broadcast system, the encryption algorithm $\mathsf{Encrypt}$ requires as an additional input a private broadcast key BK that is only known to the broadcaster.

The **length efficiency** of a broadcast encryption system is measured in the length of the header Hdr. The shorter the header, the more efficient the system. Some systems such as [BGW05, Del07, DPP07, BS03, SF07] achieve a fixed size header that depends only on the security parameter and is independent of the size of the recipient set S.

As usual, we require that the system be correct, namely that for all subsets $S \subseteq \{1, \ldots, n\}$ and all $i \in S$ if $(\mathsf{PP}, (\mathsf{SK}_1, \ldots, \mathsf{SK}_n)) \leftarrow \mathsf{Setup}(1^\lambda, n)$ and $(\mathsf{Hdr}, k) \leftarrow \mathsf{Encrypt}(\mathsf{PP}, S)$ then $\mathsf{Decrypt}(\mathsf{PP}, i, \mathsf{SK}_i, S, \mathsf{Hdr}) = k$.

Security. We define selective security for a broadcast system. Security is defined using the following experiment, denoted $\mathtt{EXP}(b)$, parameterized by the total number of recipients n and by a bit $b \in \{0, 1\}$:

$(\mathsf{PP}, (\mathsf{SK}_1, \ldots, \mathsf{SK}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$

$(S^*, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda, 1^n)$
$b' \leftarrow \mathcal{A}(\mathsf{PP}, \mathsf{state}, \{\mathsf{SK}_i\}_{i \notin S^*}, \mathsf{Hdr}, k^*)$
where
$(\mathsf{Hdr}, k_0) \leftarrow \mathsf{Encrypt}(\mathsf{PP})$, $k_1 \leftarrow \{0, 1\}^\lambda$, and $k^* \leftarrow k_b$.

For $b = 0, 1$ let W_b be the event that $b' = 1$ in $\texttt{EXP}(b)$ and as usual define $\texttt{AdvKE}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 13. *We say that a broadcast encryption system is selectively secure if for all probabilistic polynomial time adversaries $\mathcal{A}$ the function $\texttt{AdvKE}(\lambda)$ is negligible.*

Notation: Fix a set $\mathcal{Y}$, and label two elements of $\mathcal{Y}$ as 0 and 1. For a set $S \subseteq \{1, \ldots, n\}$, let $\chi(S) \in \mathcal{Y}^n$ denote a sequence of n elements of $\mathcal{Y}$ where $\chi(S)[i] = 0$ if $i \notin S$, and $\chi(S)[i] = 1$ if $i \in S$. We call $\chi(S)$ the incidence vector for S.

Construction We now construct a private key broadcast system — in Section F.2, we show how to make the scheme public key. Let F be a puncturable pseudorandom function, $\mathcal{H} : \mathcal{Y}^n \to \mathcal{Y}$ a Merkle Hash Tree, and $\mathsf{SIG} = (\mathsf{Setup}_{\mathsf{SIG}}, \mathsf{S}_{\mathsf{SIG}}, V_{\mathsf{SIG}})$ a signature scheme.

- $\mathsf{Setup}_{\mathsf{BE}}(1^\lambda, 1^n)$: The $\mathsf{Setup}_{\mathsf{BE}}$ algorithm takes the security paramemter λ and a number of users n and computes the following:
  1. Generate $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Setup}_{\mathsf{SIG}}(1^\lambda)$.
  2. Generate an instance F of a puncturable pseudorandom function with security parameter λ
  3. Compute the differing-inputs obfuscation $\mathsf{P1}_{\mathsf{obf}}$ of the program $\mathsf{P1} = \mathsf{P}^{(\mathsf{PK},F)}$, using the size of the circuit to be $\max\{|\mathsf{P}^{(\mathsf{PK},F)}|, |\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}|\}$ where $\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}$ is defined in Figure 5.
  4. For each user i, compute the signature on i: $\sigma_i \leftarrow \mathsf{S}_{\mathsf{SIG}}(\mathsf{SK}, i)$.

  It sets the public parameters, broadcast key, and user secret key as:

$$\mathsf{PP} = \mathsf{P1}_{\mathsf{obf}} \text{ and } \mathsf{BK} = F \text{ and } \mathsf{SK}_i = \sigma_i$$

- $\mathsf{Encrypt}_{\mathsf{BE}}(\mathsf{BK}, S)$. To encrypt to a set S, let $z = \chi(S) \in \mathcal{Y}^n$ be the incidence vector for S, and let $h_S = \mathcal{H}(z)$. Output an empty header, and the message encryption key $k_S = F(h_S)$.
- $\mathsf{Decrypt}_{\mathsf{BE}}(\mathsf{PP}, \sigma_i, S, h)$. To compute the message encryption key k_S, user i computes $h_S = \mathcal{H}(\chi(S))$, as well as a proof π that it knows a z such that $\mathcal{H}(z) = h$ and $z[i] = 1 \in \mathcal{Y}$. Then the message encryption key is $k_S \leftarrow \mathsf{P1}_{\mathsf{obf}}(j, \pi, i, \sigma_i)$.

$\mathsf{P}^{(\mathsf{PK},F)}$

Given input (h, π, i, σ) $\mathsf{P}^{(\mathsf{PK},F)}$ proceeds as follows:

1. Check that π is a valid proof that there exists $z \in \mathcal{Y}^n$ where $\mathcal{H}(z) = h$ and $z[i] = 1 \in \mathcal{Y}$.
2. Check that $V_{\mathsf{SIG}}(\mathsf{PK}, i, \sigma)$ accepts.
3. If any check fails, abort and output $\perp$.
4. Otherwise, output $F(h)$

Figure 4 The program $\mathsf{P}^{(\mathsf{PK},F)}$ that users will use for decryption.

Correctness. Correctness of our scheme is straightforward by inspection.

Parameter sizes. Secret keys in our scheme are just signatures, which are independent of the number of users. Headers are empty, and the public key is an obfuscations of the program in Figure 4, which only depend logarithmically on the number of users.

Security. The security of our scheme is given by the following theorem:

Theorem 4. *The scheme above is selectively secure if $\mathcal{H}$ is a collision resistant Merkle hash tree, F is a secure punctured PRF, SIG is a secure signature scheme, and the $\mathsf{P1}_{\mathsf{obf}}$ is a differing-input obfuscation of $\mathsf{P}^{(\mathsf{PK},F)}$.*

Proof. We prove security through a sequence of hybrids.
Hybrid_0: This hybrid represents the honest selective security game for broadcast encryption. The adversary commits to a set S^*. Then the adversary receives the public parameters $\mathsf{PP} = \mathsf{P1}_{\mathsf{obf}}$, and secret keys $\sigma_i = \mathsf{S}_{\mathsf{SIG}}(\mathsf{PK}, i)$ for each $i \notin S^*$. Let $h^* = \mathcal{H}(\chi(S^*))$. The adversary also receives $k^* = F(h^*)$. The adversary is now allowed to make encryption queries to any set $S \neq S^*$, to which it receives the correct message encryption key.
Hybrid_1: In this hybrid, we add the requirement that for any encryption query on a set S, that $h^* \neq \mathcal{H}(\chi(S))$. If this check fails, abort the game.
Hybrid_2: This hybrid is identical to Hybrid_1 except for the generation of $\mathsf{P1}_{\mathsf{obf}}$ in the public key. Given F, we puncture F at h^*, obtaining the program F^{h^*}. We set the public key to be the differing-inputs obfuscation $\mathsf{P2}_{\mathsf{obf}}$ of the program $\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}$ in Figure 5.

$\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}$

Given input (h, π, i, σ) $\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}$ proceeds as follows:

1. Check that $h \neq h^*$.
2. Check that π is a valid proof that there exists $z \in \mathcal{Y}^n$ where $\mathcal{H}(z) = h$ and $z[i] = 1 \in \mathcal{Y}$.
3. Check that $V_{\mathsf{SIG}}(\mathsf{PK}, i, \sigma)$ accepts.
4. If any check fails, abort and output $\perp$.
5. Otherwise, output $F^{h^*}(h)$

Figure 5 The program $\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}$ that users will use for decryption.

Hybrid_3: This hybrid is identical to Hybrid_2, except that instead of $k^* = F(h^*)$, we set k^* to be a uniform string in the codomain of F.
Hybrid_4: This is identical to Hybrid_3, except the adversary is again given the correct public key consisting of an obfuscation of $\mathsf{P}^{(\mathsf{PK},F)}$.
Hybrid_5: This hybrid is identical to Hybrid_4, except we remove the check in encryption queries that $h^* \neq \mathcal{H}(\chi(S))$. This game is identical to Hybrid_0, except that k^* is chosen at random and independent of F. Therefore, this hybrid is exactly the dishonest selective security game.

We need to argue that each of these hybrids is indistinguishable. First, if Hybrid_1 aborts during an encryption query for S, it means $\mathcal{H}(\chi(S)) = h^* = \mathcal{H}(\chi(S^*))$, and thus $\chi(S)$ and $\chi(S^*)$ form a collision for $\mathcal{H}$ (since $S \neq S^*$). By the collision resistance of $\mathcal{H}$, this can only happen with negligible probability. Therefore, Hybrid_0 is indistinguishable from Hybrid_1. The same is true of Hybrid_4 and Hybrid_5.

Our next step is to show that P and P_2 are input-indistinguishable. Indeed, suppose an adversary, given P, P_2, a set S^*, and $\sigma_i = \mathsf{S}_{\mathsf{SIG}}(\mathsf{SK}, i)$ for $i \notin S^*$ can compute an input (h, π, i, σ) where P and P_2 differ. The only way for P and P_2 to have different outputs is for P_2 to abort at a point where P does not. Thus, at any such point, it must be that $h = h^*$, π is a valid proof, and σ is a valid signature on i. There are two cases:

- $i \in S^*$. Then σ is a valid forgery. If an adversary produces such an input, we can use it to break the security of SIG. The adversary works as follows: on input PK for SIG, it

generates the programs P and P_2, and makes signature queries on $i \notin S^*$ to obtain σ_i, and gives all of these parameters to the differing-inputs adversary. The differing inputs adversary then produces a differing input $(h^*, \pi, i^*, \sigma^*)$. Since i^* is assume to be in S^*, σ^* is a valid forgery for the message i^*.

- $i \notin S^*$. Then π proves $h^* = \mathcal{H}(\chi(S'))$ for some $S' \neq S^*$. This proof gives us a collision for $\mathcal{H}$.

Therefore, assuming $\mathcal{H}$ is collision resistant and SIG is a secure signature scheme, P and P_2 are input indistinguishable. This means that the obfuscations $\mathsf{P1}_{\mathsf{obf}}$ and $\mathsf{P2}_{\mathsf{obf}}$ are indistinguishable.

Since the only difference between Hybrid_1 and Hybrid_2 is the obfuscation of two input-indistinguishable programs, the hybrids themselves are indistinguishable. The same applies to Hybrid_3 and Hybrid_4. It remains to prove that Hybrid_2 and Hybrid_3 are indistinguishable.

Suppose we have an adversary $\mathcal{A}$ distinguishing Hybrid_2 from Hybrid_3. We construct an adversary $\mathcal{B}$ breaking the security of F. $\mathcal{B}$ runs $\mathcal{A}$, and when $\mathcal{A}$ outputs a set S^*, $\mathcal{B}$ computes $h^* = \mathcal{H}(\chi(S^*))$ and asks its F challenger for the punctured PRF F^{h^*}. It also makes a challenge on h^*, obtaining the key k^*. With F^{h^*}, $\mathcal{B}$ can generate P_2, which it obfuscates and gives to $\mathcal{A}$. It also generates the parameters for SIG and gives $\mathcal{A}$ the signatures on all points not in S^*, and gives k^* as the message encryption key. If k^* is the correct key $F(h^*)$, then $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in Hybrid_2. Otherwise, the view is identical to Hybrid_3. Therefore, if $\mathcal{A}$ distinguishes Hybrid_2 from Hybrid_3 with non-negligible probability, $\mathcal{B}$ distinguishes the correct k^* from a random k^* also with non-negligible probability. The security of F therefore implies that Hybrid_2 is indisitnguishable from Hybrid_3.

We can therefore conclude that Hybrid_0 is indistinguishable from Hybrid_5, proving the security of our broadcast encryption scheme.

□

F.2 A public key broadcast scheme

In the broadcast system of the previous section the broadcaster's key BK had to be kept secret. Here we show how to modify the broadcast scheme to make it public key. Our modification is simple: we have the broadcaster generate a random input $x \in \mathcal{X}$ to a one-way function f, and let $y = f(x) \in \mathcal{Y}$. The hash h is now $\mathcal{H}(\chi(S), y)$. We we change the public program to be an obfuscation $\mathsf{P1}_{\mathsf{O}}$ of $\mathsf{P}^{(\mathsf{PK},F)}$ in Figure 6.

$\mathsf{P}^{(\mathsf{PK},F)}$

Given input (y, h, π, i, σ) $\mathsf{P}^{(\mathsf{PK},F)}$ proceeds as follows:

1. If $i = 0$:
   (a) Check that π is a valid proof that there exists $z \in \mathcal{Y}^{n+1}$ where $\mathcal{H}(z) = h$ and $z[n+1] = y$.
   (b) Check that $y = f(\sigma)$.
2. If $i \neq 0$,:
   (a) Check that π is a valid proof that there exists $z \in \mathcal{Y}^{n+1}$ where $\mathcal{H}(z) = h$, $z[i] = 1 \in \mathcal{Y}$, and $z[n+1] = y$.
   (b) Check that $V_{\mathsf{SIG}}(\mathsf{PK}, i, \sigma)$ accepts.
3. If any check fails, abort and output $\perp$.
4. Otherwise, output $F(h)$

Figure 6 The program $\mathsf{P}^{(\mathsf{PK},F)}$ that users will use for decryption.

To encrypt, the broadcaster lets $z' = \chi(S) \in \mathcal{Y}^n$, lets $z = (z', y)$, and sets $h = \mathcal{H}(z)$. The broadcaster also generates a proof π that it knows a z with $\mathcal{H}(z) = h$ and $z[n+1] = y$, and runs $\mathsf{P1_O}$ on input $(y, h, \pi, 0, x)$. The result is the message encryption key $k = F(h)$. The header is y. To decrypt, user i generates a proof π that it knows $z \in \mathcal{Y}^{n+1}$ with $z[i] = 1$, $z[n+1] = y$ and $\mathcal{H}(z) = h$, and runs $\mathsf{P1_O}$ on input (y, h, π, σ_i) to obtain the key $k = F(h)$.

For security, it is straightforward to adapt the proof from above to the public key scheme. The main difference is arguing that the program $\mathsf{P}^{(\mathsf{PK},F)}$ and the modified program $\mathsf{P}_2^{(\mathsf{PK},h^*,F^{h^*})}$ which aborts if $h = h^*$ form a differing-input circuit family. The only difference in the argument is that a differing input might have $i = 0$. But in this case, the collision resistance of $\mathcal{H}$ implies that $y = y^*$ from the challenge, and that $f(\sigma) = y^*$, which means σ is a preimage of y^*. The one-wayness of f shows that this can only happen with negligible probability, meaning P and P_2 are a differing-inputs circuit family.