# Multi-Key Searchable Encryption, Revisited

Ariel Hamlin* abhi shelat† Mor Weiss‡ Daniel Wichs§

January 5, 2018

**Abstract**

We consider a setting where users store their encrypted documents on a remote server and can selectively share documents with each other. A user should be able to perform keyword searches over all the documents she has access to, including the ones that others shared with her. The contents of the documents, and the search queries, should remain private from the server.

This setting was considered by Popa et al. (NSDI '14) who developed a new cryptographic primitive called *Multi-Key Searchable Encryption (MKSE)*, together with an instantiation and an implementation within a system called Mylar, to address this goal. Unfortunately, Grubbs et al. (CCS '16) showed that the proposed MKSE definition fails to provide basic security guarantees, and that the Mylar system is susceptible to simple attacks. Most notably, if a malicious Alice colludes with the server and shares a document with an honest Bob then the privacy of all of Bob's search queries is lost.

In this work we revisit the notion of MKSE and propose a new strengthened definition that rules out the above attacks. We then construct MKSE schemes meeting our definition. We first give a simple and efficient construction using only *pseudorandom functions*. This construction achieves our strong security definition at the cost of increasing the server storage overhead relative to Mylar, essentially replicating the document each time it is shared. We also show that high server storage overhead is not inherent, by giving an alternate (albeit impractical) construction that manages to avoid it using obfuscation.

*Northeastern University College of Computer and Information Science. hamlin.a@husky.neu.edu
†Northeastern University College of Computer and Information Science. abhi@northeastern.edu
‡Northeastern University College of Computer and Information Science. m.weiss@northeastern.edu
§Northeastern University College of Computer and Information Science. wichs@ccs.neu.edu

# Contents

# 1 Introduction

Searchable (symmetric) encryption (SSE) [SWP00, Goh03, CM05, CGKO06] allows a user to outsource her encrypted documents to a remote server. Later, she (or someone she authorizes) can send the server encrypted keyword search queries and receive the set of (encrypted) documents matching her keyword. Ideally, even a compromised server would not learn anything about the user's data or her queries. This functionality can in theory be achieved using techniques such as Oblivious RAM and Fully Homomorphic Encryption, but the efficiency overhead makes the resultant schemes largely impractical.

To allow for more practical schemes, SSE relaxes the ideal security requirement and allows the server to learn some *leakage*—namely the access pattern of which documents are returned by each query. The initial SSE definitions failed to capture natural attacks, and were revised by several follow-up works culminating in the work of Curtmula et al. [CGKO06], who gave meaningful definitions that captured the intuitive security goal. There are now constructions of SSE schemes that meet this definition and are simple, practically efficient, and updatable [KPR12, SPS14, CJJ+14]. We also note that there have been several works [NKW15, KKNO16, ZKP16] showing that the leakage provided by SSE can already be too damaging to give meaningful security guarantees in some contexts. Despite such attacks, it seems that in many cases SSE can provide meaningful security for certain data sets, even if it is imperfect.

One benefit of outsourcing data to the cloud is that it allows users to easily share data with each other. Therefore, it is natural to consider a setting where a large group of users store their individual documents, encrypted under their own keys, on a remote cloud server, where each document can be shared with an arbitrary subset of other users. As with SSE, a user should be able to perform keyword search queries over all of the documents she has access to, including both her own documents and the ones shared with her. A trivial solution is to have each user generate a new SSE key for each set of documents she wishes to share, and provide this key to the authorized group of users. However, this solution has two main drawbacks: the user must maintain many keys (one for each set of documents shared with her), and the query size scales with the number of documents that have been shared with the user. These limitations are undesirable in many realistic scenarios, since a user may have tens of thousands of document sets shared with her.

To avoid these drawbacks, Popa et al. [PZ13, PSV+14] introduced the notion of *Multi-Key Searchable Encryption* (MKSE) to specifically address the query size problem. They provided a formal definition of MKSE along with a construction using bilinear maps, and an implementation of their scheme within a framework called *Mylar* for building secure web-applications over encrypted data. As part of the framework, they provide a number of prototype applications, including a chat room and a medical application.

The MKSE definition of [PZ13, PSV+14] aimed at capturing the following intuitive security guarantee: the scheme hides the content of *both* queries and stored documents, and the only information leaked is whether a given query matched any of the keywords in a given document. This should hold even when a subset of corrupted users colludes with the server. However, Grubbs et al. [GMN+16] showed that Mylar does not achieve this goal, and suffers from several serious security deficiencies, far beyond the limited leakage inherent in SSE. While some of these issues only apply to the particular design choices of Mylar and its applications, others are more general problems with the proposed MKSE definition. Recently, Van Rompay et al. [RMÖ17] designed a different attack that pointed to another problem with the proposed MKSE definition. All these deficiencies remain present in follow up works that build on top of the MKSE definition of [PZ13, PSV+14], such as the work of Kiayias et al. [KOR+16]. We outline the three main issues below.

**Separating Data and Query Privacy.** The first issue with the MKSE definition is that it *separately* defines *data privacy* and *query privacy*, although it is intuitively clear that data and query privacy are *inherently* intertwined. Indeed, the server learns which documents are returned in response to a query, so knowing the contents of these documents leaks information about the contents of the query, and vice versa. Therefore, the two properties cannot be meaningfully defined in isolation. This observation has already been made in the context of single-key SSE by Curtmula et al. [CGKO06], who showed that earlier definitions that separated data and query privacy did not meaningfully rule out trivially insecure schemes. The work of [GMN+16] gives analogous examples in the context of MKSE, showing that there are trivially insecure schemes which satisfy the proposed definition.

**Malicious Users Sharing Data.** The second issue with the MKSE definition is more subtle. If an honest user Alice shares her document with a malicious user Mallory, then clearly privacy of her document is inherently lost. This limitation is intuitive, and users know they *should not* share their data with people they do not trust. But if Mallory shares her document with an honest Bob, one does not (and should not) expect Bob's security to be compromised. Unfortunately, [GMN+16] show that the proposed MKSE notion of [PZ13, PSV+14] does not guarantee any privacy for Bob's search queries in this scenario. In particular, the security game designed to capture query privacy in this setting [PZ13, Def. 5.6] *explicitly* prevents the adversary from sharing documents with the honest user (whose queries the adversary is trying to learn). Not only is this issue overlooked in the definition, but it is actually inherent to the proposed MKSE syntax, so every construction which realizes this syntax (e.g., the follow-up works of [KOR+16] and [Tan14]) *necessarily* inherits this flaw. According to the original MKSE definition, when Mallory shares a document with Bob, a *share key* $\Delta$ is generated. The share key $\Delta$ does not depend on Mallory's set, and allows *any* query under Bob's key to be transformed into a query under Mallory's key. Therefore, a malicious server, colluding with Mallory, can use $\Delta$ to transform every query Bob makes into a query under Mallory's key, and the transformed query can then be executed offline against a set of single-word documents containing the full dictionary. Thus, if Mallory shares a document with Bob, the server can (through this offline dictionary attack) recover *all keywords* Bob searched for.

**Searching by Comparing Queries and Encrypted Keywords.** The third issue is that the MKSE definition implicitly restricts the algorithmic structure of the scheme to encrypt each keyword in the document separately, and search in a document by comparing the given query to each of the encrypted keywords. Thus, a "hit" reveals not only that the query appears in the document, but also *which (encrypted) keyword it matched.* Van Rompay et al. [RMÖ17] show that this allows the server to compare queries issued by different users (even if both users are honest), and encrypted keywords from different documents (when they match the same keyword token).

### 1.1 Our Contribution

In this work, we propose a new MKSE definition that does not suffer from the above issues. In particular, our definition *simultaneously* addresses data and query privacy in a holistic manner, explicitly considers malicious data owners that may share data with honest users, and prevents the adversary from comparing queries issued by different users and keywords from different documents. We then propose a simple construction which provably satisfies our definition using only Pseudo-Random Functions (PRFs). Queries in this scheme consist of a single PRF image, and searching in a document is constant-time, but the server storage overhead is high. In particular, each time a document is shared with a user, it is essentially replicated, causing the per-document server storage overhead to be linear in the number of users the document is shared with.

We initially conjectured that such overhead is inherent to achieving our stronger MKSE definition. However, we show that proving such a conjecture will be quite challenging, since it will require ruling out the existence of certain program obfuscators. Concretely, in Section 5, we construct an MKSE scheme that uses obfuscation (specifically, public-coin differing-input obfuscation [IPS15]) and requires per-document server storage that is roughly the document size *plus* the number of users it is shared with (The scheme has constant query size and polynomial time search.) We view our construction as providing evidence that a more efficient construction may possibly achieve the stronger MKSE notion with optimal server storage overhead.

**Overview of Our MKSE Definition.** We consider users that can take on two types of roles: data owners and queriers. Data owners have a document they wish to share with some subset of the users. Each document has its own associated *data key* $K^d$, where the data owner "encrypts" the document using this key, and uploads the encrypted document to the server. Each user has a *query key* $K^u$ that it uses to issue search queries. When a data owner shares a document $d$ with a user $u$ they create a *share key* $\Delta_{u,d}$ which depends on the keys $K^u, K^d$, as well as the encrypted document, and store $\Delta_{u,d}$ on the server. When a querier wants to search for some keyword, he "encrypts" the keyword using his query key $K^u$, and sends the resulting encrypted query to the server. For each document $d$ that was shared with the user $u$, the server uses the share key $\Delta_{u,d}$ to execute the encrypted query over the encrypted document, and learns if the keyword is contained in that document. This allows the server to return all relevant documents the querier has access to and which contain the keyword.

The main syntactic difference between our notion, and the MKSE notion used in Mylar, is in how the share key $\Delta_{u,d}$ is generated. As noted above, the share key in Mylar depends only on the keys $K^u, K^d$, whereas in our notion it also depends on the encrypted document. By tying the share key to the document, we can ensure that each query can only be executed on *the specific documents that were shared with the querier*, rather than on arbitrary documents, *even if the server has the key* $K^d$.

To define security, we consider a *share graph* between data owners (documents) and queriers, representing who shares data with whom, where some subset of data owners are malicious and collude with the server. The desired security guarantee is that the server learns nothing about the contents of the documents belonging to the honest data owners, or the keywords being queried, beyond the *access pattern* of which documents are returned by each query (i.e., out of the documents shared with the querier, which ones contain the queried keyword). We provide an *indistinguishability-based* definition where the adversary chooses the documents and data keys belonging to the malicious data owners, and two potential values (a "left" and a "right" value) for each query and each document belonging to an honest data owner. The left and right values must lead to the same access pattern, and the queries of each querier must be distinct. The adversary then gets all encrypted documents, share keys, and encrypted queries, and should not be able to distinguish whether these were created using the left or right values.

Since the adversary only learns the access pattern of which documents are returned by each query, the above definition captures the *minimal* leakage for schemes that reveal the access pattern, which seems to be the case in all practical schemes. This is a significant qualitative improvement over the leakage allowed by the previous definition of [PZ13] and the corresponding schemes. Most importantly, when a malicious user Mallory is colluding with the sever and shares some data with Bob, the previous schemes completely leaked the contents of Bob's query wheres our definition still only reveals the access pattern. We note that similar to single-key SSE, leaking the access pattern does reveal some potentially sensitive information and in some scenarios (e.g., when combined with auxiliary information about the documents) this may allow a sufficiently powerful attacker to completely recover the query, as shown in the single-key SSE setting by the recent works [CGPR15, NKW15, PW16, KKNO16, ZKP16, GSB+17]. In the multi-key setting this might be am-

plified since, whenever malicious data owners share documents with honest querier, the adversary already knows (and even chooses) the contents of these documents and hence can learn more information by seeing which of these documents match the query. For example, if the shared documents correspond to every individual word in a dictionary, then by seeing which document matches a given query the contents of the query are completely revealed. However, this is not a very natural scenario, and in many settings it is reasonable to believe that leaking the access pattern alone may not reveal significant information about the query. Furthermore, users can perform sanity checks on the documents shared with them to test how much leakage the server will get on their queries, and refuse to accept shared documents if they lead to too much leakage. Understanding when access pattern leakage is acceptable and when it is not is a fascinating and important direction for future study.

One implementation concern in the above notion of MKSE comes from how the share key $\Delta_{u,d}$ is generated, since it relies on knowledge of both the data-owner's key $K^d$ for the document being shared, the user's querier key $K^u$, and the encrypted document itself. We envision that the data owner simply sends the data key $K^d$ to the querier via a secure channel. The querier then downloads the encrypted document from the server, generates $\Delta_{u,d}$, and uploads it to the server. Note that the querier can also check at this point that the document was encrypted correctly, and therefore in the security definition we always assume that documents are encrypted honestly.

Finally, our default definition is *selective*, meaning the adversary specifies the entire share graph, the data, and the queries ahead of time. We can also consider *adaptive* security for SSE (a notion introduced by [CGKO06] in the single-user setting) in which the adversary generates queries on the fly during the course of the attack. Furthermore, our definition is *indistinguishability based*, where one could also consider a *simulation-based* version (as introduced in the single-user setting by [CGKO06]) in which the simulator, given the share graph, document sizes, and access patters, produces the encrypted documents and queries. We discuss these alternate variants in Section 6, and note that our PRF-based construction described below satisfies the strongest security notion (adaptive, simulation-based) when the PRF is instantiated in the random-oracle model.

**Overview of the PRF-based Construction.** We provide a simple and efficient MKSE scheme based only on the existence of one-way functions. As noted above, each share key $\Delta_{u,d}$ contains a copy of the document, which allows Search to use only this value (and not the encrypted document).

If $\Delta_{u,d}$ is "allowed" to encode the entire document, a natural approach is to assign to each querier a PRF key $K^u$ for a PRF $F$, and store in $\Delta_{u,d}$ the images of $F(K^u, \cdot)$ on all keywords in the document. However, this construction is fundamentally insecure, since the share keys themselves leak information, *even if the querier never makes any queries*, and *even if all data owners are honest*. More specifically, consider the case of two honest data owners that share their documents with an honest querier. Then the two share keys reveal the number of keywords that appear in both documents. This is because the token associated with each keyword depends only on the keyword and the querier key, and *not* on the document.

To solve this issue we use another layer of PRF images, where the first layer generates PRF keys for a second layer that will be applied to a document-specific random identifier. Concretely, when generating $\Delta_{u,d}$, we assign a random value $r$ to the document. For every keyword $w$ in the document, we generate a (second layer) PRF key $k^w = F(K^u, w)$, and compute a token $t^w$ for $w$ as $t^w = F(k^w, r)$. Using perfect hashing [FKS82], these tokens are inserted into a hash table to accelerate searching. The share key $\Delta_{u,d}$ consists of $r$ and the hash table containing the tokens. (Notice that if $r$ is chosen from a sufficiently large domain, then with overwhelming probability each document is assigned a *unique* identifier, and so the share keys associated with two documents reveal no information about their intersection.)

To search for keyword $w$ in her documents, the querier sends the query $k^w = F(K^u, w)$ to the

server. Searching for $k^w$ in a document with $\Delta_{u,d} = (r, D')$ (where $D'$ is a hash table of tokens) is performed by searching the hash table $D'$ on the key $F(k^w, r)$. This query reveals no information about $w$. Notice that the scheme uses the encrypted document only to generate $\Delta_{u,d}$, so the document can simply be encrypted with its own unique symmetric encryption key.

### 1.2 Related Work

**Single User Schemes.** First introduced by Song et al. [SWP00], the notion of (single user) Searchable Encryption has been extensively studied in the last decade (see [BHJP14, FVY+17] for a survey of many of these works). The first works (e.g., [SWP00, Goh03, CM05]) constructed schemes under several (simulation-based or indistinguishability-based) security definitions. These definitions separated the properties of query and data privacy, and were shown by [CGKO06] to be insecure (by a fairly simple attack). Curtmola et al. [CGKO06] also presented a unified definition that combined both properties.

**Multi-User Schemes.** In this model multiple users can issue queries to a single dataset which is encrypted under a single key that is known to all users. Consequently, most works in this setting focus on access control, and efficient revocation of querying privileges. Following the work of [CGKO06], who provided the first definitions and constructions, there have been three main approaches to enforcing access control across the documents: traditional access control mechanisms [DRD11, ZNS11, LLC+12], broadcast encryption [CGKO06, LWC+13], and attribute-based encryption [CLH+14, LCL+14].

We emphasize that in the multi-*user* setting, there is a *single* dataset owned by a single data owner, so using such schemes in settings with multiple data owners would require instantiating the scheme separately for each dataset, and thus the query size would be *linear* in the number of datasets shared with the querier. This should be contrasted with the multi-*key* setting which is the focus of this work, in which users can search over multiple datasets by issuing a *single* query whose size is *independent* of the number of datasets being searched.

**Multi-Key Schemes.** In this setting multiple users share data encrypted under their own keys, and search across the data shared with them by issuing a single query whose size is independent of the number of shared datasets. First introduced by Popa et al. [PSV+14], follow-up works that build on [PSV+14] focused on optimizing server storage, and eliminating the trusted party needed to distribute data and querier keys [KOR+16]; mitigating attacks in which a malicious data owner shares a dictionary dataset with the querier, by having the querier explicitly determine which data owners are allowed to share data with her [Tan14][1]; and constructing schemes that are secure in restricted security models when honest data owners only share their documents with honest queriers, or when a single data owner has not shared his dataset with anyone else [Tan14] (in both models, the server might be corrupted). We note that since these works use the syntax of [PSV+14] (in particular, share keys are generated independently of the shared set), the aforementioned attacks of [GMN+16] apply to these works as well.

**Other Related Models.** The notion of Key Aggregate Searchable Encryption (KASE), introduced by [CLW16], considers a data owner who has several documents, each encrypted under a unique key. This allows data owners to share different subsets of their documents with different users. The goal is to grant search access to a subset of documents by providing *one* aggregate key whose length is independent of the number of documents (whereas in a naive solution, the key size would scale with the number of documents), and the querier issues *one* query for *every* subset of documents shared

[1] This functionality was also discussed in [PSV+14], but was not defined as part of the MKSE syntax.

under the aggregate key (whereas in the MKSE setting, the user issues *a single query*, *regardless* of the number of documents shared with her). Thus, this model is fundamentally different from MKSE (as pointed out in [CLW16]). We note that the construction of [CLW16] is vulnerable to dictionary attacks (as shown by [KOR+16]).

## 2 Preliminaries

In the following, $\lambda$ denotes a security parameter, and $\mathsf{negl}(\lambda)$ denotes a function that is negligible in $\lambda$. We use $\approx$ to denote computational indistinguishability, and $S \setminus T$ to denote the difference between sets $S, T$. We use $\Pr[E : E_1, \cdots, E_n]$ to denote the probability of event $E$ given events $E_1, \cdots, E_n$. For strings $x = x_1 \cdots x_n, y = y_1 \cdots y_m$, $x \circ y$ denotes their concatenation, i.e., $x \circ y = x_1 \cdots x_n y_1 \cdots y_m$. We use standard cryptographic definitions of one-way functions (OWFs), one-way permutations (OWPs), collision resistant hash functions (CRHFs), pseudorandom functions (PRFs), and existentially-unforgeable signature schemes (see, e.g., [Gol01, Gol04]).

## 3 Defining Multi-Key Searchable Encryption

In this section we define the notion of *Multi-Key Searchable Encryption* (MKSE) schemes. Intuitively, an MKSE scheme allows data owners to share their documents with queriers who can later query these documents under their own keying material, while preserving both *data* and *query privacy*. In the definition, documents are represented as *sets of keywords*, so searching in a document translates to checking set membership; see the discussion following the definition.

**Definition 3.1** (Multi-key Searchable Encryption)**.** We say that a tuple $(\mathsf{DataKeyGen}, \mathsf{QueryKeyGen}, \mathsf{ProcessSet}, \mathsf{Share}, \mathsf{Query}, \mathsf{Search})$ of PPT algorithms is a *Multi-key Searchable Encryption (MKSE)* scheme for a universe $\mathcal{U}$, if the following holds.

- **Syntax:**
  - $\mathsf{DataKeyGen}$ takes as input the security parameter $1^\lambda$, and outputs a data key $K$.
  - $\mathsf{QueryKeyGen}$ takes as input the security parameter $1^\lambda$, and outputs a query key $K^u$.
  - $\mathsf{ProcessSet}$ takes as input a data key $K$ and a set $S$, and outputs a processed set $T$.
  - $\mathsf{Share}$ takes as input a data key $K$, a query key $K^u$, and a processed set $T$, and generates a user-specific share key $\Delta$.
  - $\mathsf{Query}$ takes as input an element $w \in \mathcal{U}$ and a query key $K^u$, and outputs a query $q$.
  - $\mathsf{Search}$ takes as input a user-specific share key $\Delta$, a query $q$, and a processed set $T$, and outputs $b \in \{0, 1\}$.
- **Correctness:** For every security parameter $\lambda \in \mathbb{N}$, data set $S \subseteq \mathcal{U}$, and element $w \in \mathcal{U}$:

$$\Pr\left[\mathsf{Search}(\Delta, q, T) = b \ : \ \begin{array}{rl} K & \leftarrow \mathsf{DataKeyGen}\left(1^\lambda\right) \\ K^u & \leftarrow \mathsf{QueryKeyGen}\left(1^\lambda\right) \\ T & \leftarrow \mathsf{ProcessSet}(K, S) \\ \Delta & \leftarrow \mathsf{Share}(K, K^u, T) \\ q & \leftarrow \mathsf{Query}(K^u, w) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

  Where $b = 0$ if $w \notin S$, otherwise $b = 1$.

- **Security:** Every PPT adversary $\mathcal{A}$ has only a $\mathsf{negl}(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$:

  1. $\mathcal{A}$ sends to $\mathcal{C}$:
     - A set $\mathcal{Q} = \{1, \ldots, m\}$ of queriers, a set $\mathcal{D} = \{1, \ldots, n\}$ of data owners, and a subset $\mathcal{D}_c \subseteq \mathcal{D}$ of corrupted data owners.
     - For every $i \in \mathcal{D}_c$, a data key $K_i$.
     - For every $i \in \mathcal{D}$, two sets $S_i^0, S_i^1 \subseteq \mathcal{U}$, where $\left|S_i^0\right| = \left|S_i^1\right|$ for $i \notin \mathcal{D}_c$, and $S_i^0 = S_i^1$ for $i \in \mathcal{D}_c$.
     - A bipartite share graph $G = (\mathcal{Q}, \mathcal{D}, E)$.
     - For every $j \in \mathcal{Q}$, two sequences of distinct keywords $\left(w_{j,1}^0, \ldots, w_{j,k_j}^0\right)$ and $\left(w_{j,1}^1, \ldots, w_{j,k_j}^1\right)$ (for some $k_j \in \mathbb{N}$), such that for every $i \in \mathcal{D}$, if $(j,i) \in E$ then for every $1 \leq l \leq k_j$, $w_{j,l}^0 \in S_i^0$ if and only if $w_{j,l}^1 \in S_i^1$.
  2. $\mathcal{C}$ performs the following:
     - Chooses a random bit $b \leftarrow \{0,1\}$.
     - For each querier $j \in \mathcal{Q}$, generates $K_j^u \leftarrow \mathsf{QueryKeyGen}\left(1^\lambda\right)$.
     - For each data owner $i \in \mathcal{D} \setminus \mathcal{D}_c$, generates $K_i \leftarrow \mathsf{DataKeyGen}\left(1^\lambda\right)$.
     - For each set $S_i^b, i \in \mathcal{D}$, generates $T_i \leftarrow \mathsf{ProcessSet}\left(K_i, S_i^b\right)$.
     - For each edge $(j,i) \in E$, generates $\Delta_{j,i} \leftarrow \mathsf{Share}\left(K_i, K_j^u, T_i\right)$.
     - For each querier $j$ and keyword $w_{j,l}^b, 1 \leq l \leq k_j$, generates a query $q_{j,l} \leftarrow \mathsf{Query}\left(K_j^u, w_{j,l}^b\right)$.
     - Sends $\{T_i\}_{i \in \mathcal{D}}$, $\{\Delta_{j,i}\}_{(j,i) \in E}$, and $\left(q_{j,1}, \cdots, q_{j,k_j}\right)_{j \in \mathcal{Q}}$ to $\mathcal{A}$.
  3. $A$ outputs a guess $b'$, and its advantage is $\mathsf{Adv}_{\mathcal{A}}\left(1^\lambda\right) = \frac{1}{2} - \Pr\left[b = b'\right]$.

**Discussion.** In Definition 3.1, *sets of keywords* are shared with queriers, and share keys are generated by an honest party. Such schemes can be easily adapted to the setting in which *documents* are shared between users: each document $d$ is associated with the set $S$ of keywords it contains; and an encryption of $d$ is stored alongside the processed set $T_S$. Searching for keywords in $d$ is performed by searching for the keyword in $S$, where if the search outputs 1 then the encryption of $d$ is returned to the querier. Moreover, a trusted party is not needed to generate the share keys: we envision that each user will generate her own share keys whenever a document is shared with her. More specifically, when a data owner $i$ shares a document $d_i$ with a querier $j$, the data owner will send his data key $K_i$ to the querier via a secure channel. The querier will then download the processed set $T_i$ and the encryption of $d_i$ from the server, and generate $\Delta_{j,i}$ herself. This use case also clarifies our assumption that sets are honestly processed: $j$ can verify that $T_i$ was honestly generated by processing $S_i$ (which can be extracted from $d_i$) using $K_i$, and comparing to $T_i$. (Without loss of generality, $\mathsf{ProcessSet}$ is deterministic since any randomness can be provided in $K_i$.)

Notice that in our definition, the share key is (syntactically) "tied" to the set for which it was generated ($\mathsf{Share}$ depends not only on the data and query keys, but also on the processed set). This should be contrasted with the syntax used in [PZ13, PSV+14] in which the algorithm generating the share keys depends *only* on the data and query keys. Consequently, resultant schemes *inherently* guarantee no query privacy when malicious data owners share their sets with honest queriers (as discussed in the introduction). Indeed, when $\Delta$ is independent of the set then a malicious server

colluding with the data owner can use the data key $K$ to encrypt *any* set $S$ of his choosing (in particular, a dictionary), then use $\Delta$ to search for the query in $S$. Since $K$ was generated independently of the set, the correctness of the scheme guarantees that the output of search will be correct, and so the server can recover the queried keyword.

Similar to previous works in the field, our definition allows for some information leakage. Specifically, since the adversary is restricted to choosing sets and queries for which $w^0_{j,l} \in S^0_i \Leftrightarrow w^1_{j,l} \in S^1_i$ for every $(j,i) \in E$ (see the last bullet in Step 1 of the security game), the scheme leaks the access pattern of which subset of documents is returned in response to each query. Additionally, since we require that each querier makes distinct queries, if a querier makes repeated queries then this might be leaked to the server. Finally, we note that our definition is *selective*: the adversary is required to specify in advance the sets, queries, and share graph. Possible extensions and generalizations include adaptive security, where the adversary can adaptively choose sets and queries, add edges to the share graph, and corrupt data owners; and simulation-based security, which guarantees that the view of every PPT adversary can be simulated given only the aforementioned leakage (namely, the access patterns and the sizes of the sets). We elaborate on these alternative definitions in Section 6.

# 4 MKSE with Fast Search

In this section we describe our MKSE scheme based on PRFs. Concretely, we will prove the following theorem.

**Theorem 4.1** (MKSE (Sublinear Search))**.** *Assume that OWFs exists. Then there exists a secure MKSE scheme in which searching for keywords in a set $S$ takes* $\mathsf{poly}(\lambda)$ *time, where $\lambda$ is the security parameter.*

*Moreover, data and query keys, as well as queries, have length* $\mathsf{poly}(\lambda)$*, and for a set $S$, its processed version and share keys have size* $|S| \cdot \mathsf{poly}(\lambda)$.

We first describe our construction, then analyze its properties.

**Construction 1** (MKSE (Sublinear Search))**.** The construction uses a PRF $F$, and a symmetric-key encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, as building blocks.

- $\mathbf{DataKeyGen}\left(1^\lambda\right)$ outputs a symmetric key $K_{\mathsf{SE}} \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$.
- $\mathbf{QueryKeyGen}\left(1^\lambda\right)$ outputs a uniformly random PRF key $K_{\mathsf{PRF}} \leftarrow \{0,1\}^\lambda$.
- $\mathbf{ProcessSet}\left(K_{\mathsf{SE}}, \mathcal{S}\right)$ outputs $\mathsf{Enc}_{K_{\mathsf{SE}}}(S)$.
- $\mathbf{Share}\left(K_{\mathsf{SE}}, K_{\mathsf{PRF}}, T = \mathsf{Enc}_{K_{\mathsf{SE}}}(S)\right)$ operates as follows:
  - Generates a uniformly random string $r \leftarrow \{0,1\}^\lambda$.
  - Decrypts $S \leftarrow \mathsf{Dec}_{K_{\mathsf{SE}}}(T)$.
  - Initializes $D = \emptyset$. For each keyword $w_i \in S$, computes $k'_i = F_{K_{\mathsf{PRF}}}(w_i)$ and $d_i = F_{k'_i}(r)$, and adds $d_i$ to $D$.
  - Insert $D$ into a perfect hash table [FKS82] to obtain $D'$.
  - Outputs $\Delta = (r, D')$.
- $\mathbf{Query}\left(K_{\mathsf{PRF}}, w\right)$ outputs $F_{K_{\mathsf{PRF}}}(w)$.
- $\mathbf{Search}\left(\Delta = (r, D'), q, T\right)$ operates as follows:

– Computes $d' = F_q(r)$.
– Performs a hash table query on $D'$ for $d'$, and outputs 1 if and only if $d'$ was found.

The next claim states that Construction 1 is secure, and summarizes its parameters.

**Claim 4.2.** *Assume that Construction 1 is instantiated with a PRF $F$, and a secure symmetric encryption scheme, then it is a secure MKSE scheme.*

*Moreover, data and query keys have length $\mathsf{poly}(\lambda)$, and for a set $S$ the processed set has size $|S| \cdot \mathsf{poly}(\lambda)$. Furthermore, searching in a set $S$ takes time $\mathsf{poly}(\lambda)$ and queries have size $\mathsf{poly}(\lambda)$.*

*Proof.* The correctness of the scheme follows directly from the correctness of the underling primitives, and its complexity follows directly from the construction and from the following theorem due to Fredman et al [FKS82].

**Theorem 4.3** (Perfect hashing [FKS82])**.** *Given a set $D$ of $n$ keys from a universe $\mathcal{U}$, there exists a method that runs in expected $O(n)$ time and constructs a lookup table $D'$ of size $O(n)$ such that membership queries (i.e., given $x \in \mathcal{U}$, determine if $x \in S$) can be answered in constant time.*

We now argue that the scheme is secure.

For every $i \in \mathcal{D}$, let $S_i^0, S_i^1$ be the sets $\mathcal{A}$ chose for data owner $i$, and let $\mathcal{W}_j^0, \mathcal{W}_j^1$ be the sets of queries $\mathcal{A}$ chose for querier $j \in \mathcal{Q}$. Let $F^1$ denote the PRF called in $\mathsf{Query}$ and to compute $k_i'$ in $\mathsf{Share}$, and let $F^2$ be the PRF invoked to compute $d_i$ in $\mathsf{Share}$. Let $\mathsf{view}_0, \mathsf{view}_1$ denote the view of $\mathcal{A}$ in the security game when $b = 0, 1$ (respectively). We show that $\mathsf{view}_0 \approx \mathsf{view}_1$ using a sequence of hybrid distributions, and conclude that $\mathcal{A}$ has only a $\mathsf{negl}(\lambda)$ advantage in the security game. As the data keys, and encrypted sets, of corrupted data owners are identically distributed in both views (because $S_i^0 = S_i^1$ for every $i \in \mathcal{D}_c$), we can fix these values into $\mathsf{view}_0, \mathsf{view}_1$, and all hybrid distributions, without decreasing the computational distance. Moreover, if $F$ is sufficiently expanding then with overwhelming probability, all images of the form $F_K^1(w)$, and $F_{k_i'}^2(r)$ (for query keys $K$, keywords $w$, set identifiers $r$, and $k_i'$) are distinct, so it suffices to bound the computational distance conditioned on this event. We now define the hybrids.

For $b \in \{0,1\}$, let $\mathcal{H}_0^b$ be the distribution obtained from $\mathsf{view}_b$ by replacing $F^1$ with a random function $\mathcal{R}$. (We think of $\mathcal{R}$ as taking two inputs, the first being a querier index. Thus, $\mathcal{R}$ defines a *family* $\{\mathcal{R}_j\}_j$ of functions, as does $F^1$.) That is, for all queriers $j$, and $w_l \in \mathcal{W}_j^b$, we have $q_{j,l}' = \mathcal{R}(w_i)$ (the tag is used to denote queries in $\mathcal{H}_0^b$; queries in $\mathsf{view}_b$ are untagged). Then $\mathsf{view}_b \approx \mathcal{H}_0^b$ follows from the pseudorandomness of $F$ by a standard hybrid arguments in which we replace the invocations of $F^1$ (used to generate the queries and share keys) of one querier at a time.

We now define $\mathcal{H}_1^b$ to be identical to $\mathcal{H}_0^b$, except that $F^2$ is replaced with the random function $\mathcal{R}$ (notice that here, the first input of $\mathcal{R}$ corresponds to a query $q'$), and the keyword tokens in every share key $\Delta_{j,i}$ are generated as follows. For every $w_l \in S_i^b \cap \mathcal{W}_j^b$, the corresponding token $d_{i,j,l}$ is computed as $F_{q_{j,l}'}^2(r)$ (i.e., identically to how it is generated in $\mathcal{H}_0^b$; this is needed since $q_{j,l}'$ appears in $\mathcal{H}_1^b$ and so consistency of these tokens with $F^2$ can be efficiently checked). For every $w_l \in S_i^b \setminus \mathcal{W}_j^b$, $d_{i,j,l}'$ is chosen randomly subject to the constraint that $d_{i,j,l}' \notin \left\{ F_{q_{j,l'}'}^2(r) \ : \ w_{l'} \in \mathcal{W}_j^b \right\}$. (This can be efficiently achieved by re-sampling, assuming $F$ is sufficiently stretching.) All values in $\Delta_{j,i}$ are then hashed.

To show that $\mathcal{H}_0^b \approx \mathcal{H}_1^b$, we first define an intermediate hybrid $\mathcal{H}^{b,\star}$ in which for every querier $j$, every data owner $i$, and every keyword $w_l \in S_i^b \setminus \mathcal{W}_j^b$, the token $d_{i,j,l}'$ in $\Delta_{j,i}$ is replaced with a random value, subject to the constraint that it is not in $\left\{ F_{q_{j,l'}'}^2(r) \ : \ w_{l'} \in \mathcal{W}_j^b \right\}$, where $r$ is the

random identifier associated with $\Delta_{j,i}$. Then $\mathcal{H}_0^b \approx \mathcal{H}^{b,\star}$ follows from the pseudorandomness of $F$ by a standard hybrid argument in which we replace the tokens one at a time (and use the assumption that all images of $F$ are unique).

To show that $\mathcal{H}_1^b \approx \mathcal{H}^{b,\star}$, we define a series of sub-hybrids, replacing $F^2$ with a random function for a single query of a single querier at a time. (Notice that each query of each querier represents a unique key for $F^2$ in all share keys associated with that querier.) Concretely, denote $m = |\mathcal{Q}|$, and for every $j \in \mathcal{Q}$, let $l_j := \left|\mathcal{W}_j^b\right|$. For every $1 \leq j \leq m$ and $0 \leq l \leq l_j$, define $\mathcal{H}^{b,j,l}$ to be the distribution obtained from $\mathcal{H}^{b,\star}$ by generating the queries of the first $j-1$ queriers, and the first $l$ queries of querier $j$, with $\mathcal{R}$ (instead of $F^2$), and generating the keyword tokens in share keys accordingly. Then $\mathcal{H}^{b,1,0} = \mathcal{H}^{b,\star}$, and $\mathcal{H}^{b,m,l_m} = \mathcal{H}_1^b$. For every $1 \leq j \leq m$ and $1 \leq l \leq l_j$, $\mathcal{H}^{b,j,l} \approx \mathcal{H}^{b,j,l-1}$ by the pseudorandomness of $F^2$. Moreover, $\mathcal{H}^{b,j,0} = \mathcal{H}^{b,j-1,l_{j-1}}$ for every $1 < j \leq m$, so $\mathcal{H}^{b,1,0} \approx \mathcal{H}^{b,m,l_m}$ (since $m = \mathsf{poly}(\lambda)$, and $l_j = \mathsf{poly}(\lambda)$ for every $j \in \mathcal{Q}$).

Finally, let $\mathcal{H}_2^b$ be identical to $\mathcal{H}_1^b$ except that the encrypted sets of all honest data owners $i \notin \mathcal{D}_c$ encrypt $\vec{0}$ (instead of $S_i^b$). Then $\mathcal{H}_1^b \approx \mathcal{H}_2^b$ follows from the security of the encryption scheme by a standard hybrid argument in which the encrypted sets are replaced one at a time. Notice that $\mathcal{H}_2^0 = \mathcal{H}_2^1$ and so $\mathsf{view}_0 \approx \mathsf{view}_1$. □

**Remark 4.4.** Notice that $\mathcal{H}_2^b$ depends only on the share graph, the sets of corrupted data owners, the *sizes* of sets of honest data owners, and the access patterns; and can be efficiently generated given these values. This implies that the view of every PPT adversary can be efficiently simulated given only these values, namely, Construction 1 is *simulation*-secure (as defined in Section 6).

The proof of Theorem 4.1 now follows as a corollary from Claim 4.2.

*Proof of Theorem 4.1.* We instantiate Construction 1 with any sufficiently stretching PRF (e.g., $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^{2(\lambda+n)}$, whose existence follows from the existence of OWFs), and a secure symmetric encryption scheme (which can be constructed from $F$). Then the security of the scheme, as well as the length of data keys, query keys, and processed sets, follow directly from Claim 4.2. As for search time, since keywords have length $O(\lambda)$ then evaluating $F$ takes $\mathsf{poly}(\lambda)$ time, and the outputs have length $\mathsf{poly}(\lambda)$. Searching in a set $S$ takes 1 hash query and thus the overall time is $\mathsf{poly}(\lambda)$ time, and queries have length $\mathsf{poly}(\lambda)$. □

# 5 MKSE with Short Share Keys

In this section we describe an MKSE scheme with short share keys which employs a program obfuscator as a building block. We first show (Section 5.1) a scheme based on differing-inputs obfuscation (diO), then show (Section 5.2) that a slightly modified construction can be based on *public-coin* differing-inputs obfuscation (pc-diO). We note that though there is evidence that *diO* for general circuits might not exist [GGHW14, BSW16], no such implausibility results are known for *pc*-diO. The relevant definitions appear in Appendix A.

## 5.1 MKSE from Differing-Inputs Obfuscation

We construct a secure MKSE scheme which employs a diO obfuscator for Turing Machines (TMs). Concretely, we prove the following for a universe $\mathcal{U}$ of size $|\mathcal{U}| \leq \mathsf{poly}\left(2^\lambda\right)$:

**Theorem 5.1** (MKSE (Short Share Keys))**.** *Assume that CRHFs, and diO for TMs with polynomial blowup, exist. Then there exists a secure MKSE in which share keys have size* $\mathsf{poly}(\lambda)$*, where* $\lambda$ *is a security parameter. Moreover, data and query keys, as well as queries, have length* $\mathsf{poly}(\lambda)$*, and given a set* $S$*, its processed version has size* $|S| \cdot \mathsf{poly}(\lambda)$*, and searching in it takes* $\mathsf{poly}(\lambda, |S|)$ *time.*

The high-level idea of the constructions is to encrypt sets under their data key, and queries under the query key of the querier, using a standard (symmetric) encryption scheme. The share key will be an obfuscation of a program that has both keys hard-wired into it, and thus allows for searching (even though queries and sets are encrypted under different keys) by decrypting the ciphertexts and comparing the underlying keywords. However, to make this rough intuition work, we need to handle a few subtleties.

First, to obtain security, the program should take *the entire set* as input. Otherwise (i.e., if it operates on a single set element at a time), its output would reveal not only *whether* the queried keyword appears in the set, but also *where* it appears. To see why this violates security, consider the case in which the same set $S_i$ is shared with two different queriers $j$ and $j'$: the additional information of where in the set a query appears allows the server to check whether $j$ and $j'$ queried the same keyword (even when $j, j'$ and data owner $i$ are all honest). Notice that since the program takes the entire set as input, it cannot be represented as a circuit (since then share keys will not have sublinear size). Therefore, we implement the program as *a TM*, and use an obfuscator for TMs.

Second, as noted in the introduction, share keys should only allow searching for keywords *in the sets for which they were generated*. That is, a share key $\Delta_{j,i}$ between querier $j$ and data owner $i$ should be "tied" to the set $S_i$ of $i$. We achieve this by hard-wiring a hash $h(S_i)$ of $S_i$ into the program $P_{j,i}$ obfuscated in $\Delta_{j,i}$, where $P_{j,i}$ checks that its input set is consistent with the hash. Notice that the hard-wired hash prevents us from using an indistinguishability obfuscator [BGI+01]. Indeed, if $i$ is honest then in the security game (Definition 3.1), the adversary chooses a pair $S_i^0, S_i^1$ of (possibly different) sets for $i$, and $\Delta_{j,i}$ has either $h\left(S_i^0\right)$ (in the game with $b = 0$) or $h\left(S_i^1\right)$ (in the game with $b = 1$) hard-wired into it. In particular, the underlying programs are not functionally equivalent, so we cannot rely on indistinguishability obfuscation, and need to use a stronger primitive. Concretely, our constructions rely on the existence of a diO, or a pc-diO, obfusctor. We proceed to describe the diO-based construction (the pc-diO-based construction is described in Section 5.2).

The last ingredient we need is a signature scheme, which will be used to sign queries. Specifically, a query for keyword $w$ will consist of an encryption $c$ of $w$, and a signature on $c$; and share keys will have the corresponding verification key hard-wired into them. Intuitively, signatures are used to guarantee the server can only search for queries the querier *actually* issued, similar to the way the hashed set prevents the server from searching in sets that were not shared with the querier. Concretely, the signatures guarantee that the share keys in the security game for $b = 0$ and $b = 1$ are differing-inputs even given the entire view of the adversary, and allows us to rely on diO security. (Roughly, a pair of programs are differing-inputs if it is infeasible for a PPT algorithm to find an input on which their outputs differ, see Definition A.1 in Appendix A.)

**Remark 5.2.** We note that if one is willing to change the MKSE syntax, allowing the server to return *encrypted* answers which the querier then decrypts, then a scheme with similar complexity could be constructed from Fully Homomorphic Encryption (using Oblivious RAM or Private Information Retrieval). However, following previous works in the field [PSV+14, KOR+16, GMN+16] we focus on the setting in which the server gets the answers in the clear (and queriers do not need to decrypt). This may be crucial in some situations, e.g., when huge documents are associated with small keyword sets. In a solution based on Fully Homomorphic Encryption, the computation of a search is proportional to the total size of *all the huge documents*, while in our obfuscation-based MKSE scheme the work is proportional to the total number of *keywords*, and the size of the *returned* documents.

We now describe our MKSE scheme. As will become evident from the security proof, due to the technicalities of using diO security we will need a special type of encryption (which, nonetheless, can be constructed from any standard encryption scheme) that we call *double encryption*. It is similar to the encryption scheme used in the "2-key trick" of Naor and Yung [NY90] (to convert a CPA-secure

encryption scheme into a CCA-secure one), except it does not use a non-interactive zero-knowledge proof to prove that ciphertexts encrypt the same value.

**Definition 5.3** (Double encryption)**.** Let $\lambda \in \mathbb{N}$ be a security parameter. Given a symmetric encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, we define a *double symmetric encryption* scheme $\mathsf{E}^2 = \left(\mathsf{KeyGen}^2, \mathsf{Enc}^2, \mathsf{Dec}^2\right)$ as follows:

- $\mathsf{KeyGen}^2$, on input $1^\lambda$, generates $\mathsf{K}_L \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$ and $\mathsf{K}_R \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$, and outputs $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R)$.
- $\mathsf{Enc}^2$, on input a key $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R)$ and a message $m$, computes $c_L \leftarrow \mathsf{Enc}\left(\mathsf{K}_L, m\right)$ and $c_R \leftarrow \mathsf{Enc}\left(\mathsf{K}_R, m\right)$, and outputs $c = (c_L, c_R)$.
- $\mathsf{Dec}^2$, on input a key $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R)$ and a ciphertext $c = (c_L, c_R)$, outputs $\mathsf{Dec}\left(\mathsf{K}_L, c_L\right)$. (Notice that decryption disregards the "right" component of $c$.)

We are now ready to describe our MKSE scheme.

**Construction 2** (MKSE (Short Share Keys))**.** The MKSE uses the following building blocks:

- an obfuscator $\mathcal{O}$,
- a hash function $h$,
- a double symmetric encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, and
- a signature scheme $(\mathsf{KeyGen}_s, \mathsf{Sign}, \mathsf{Ver})$,

and is defined as follows:

- $\mathsf{DataKeyGen}\left(1^\lambda\right)$ generates a random encryption key $\mathsf{K} \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$ and outputs $\mathsf{K}$.
- $\mathsf{QueryKeyGen}\left(1^\lambda\right)$ generates a random encryption key $\mathsf{K}^u \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$, and a random signing and verification key pair $(\mathsf{sk}^u, \mathsf{vk}^u) \leftarrow \mathsf{KeyGen}_s\left(1^\lambda\right)$, and outputs $K^u = (\mathsf{K}^u, \mathsf{sk}^u, \mathsf{vk}^u)$.
- $\mathsf{ProcessSet}\left(\mathsf{K}, \mathcal{S}\right)$ encrypts each element $s \in \mathcal{S}$ as $c(s) \leftarrow \mathsf{Enc}\left(\mathsf{K}, s\right)$, and outputs $\{c(s) \ : \ s \in \mathcal{S}\}$.
- $\mathsf{Share}\left(\mathsf{K}, K^u = (\mathsf{K}^u, \mathsf{sk}^u, \mathsf{vk}^u), T_S\right)$ generates $\widetilde{P} \leftarrow \mathcal{O}\left(P_{\mathsf{K},(\mathsf{K}^u,\mathsf{vk}^u),h(T_S)}\right)$ where $P_{\mathsf{K},(\mathsf{K}^u,\mathsf{vk}^u),h(T_S)}$ is the TM defined in Figure 1, and outputs $\Delta = \widetilde{P}$.
- $\mathsf{Query}\left(K^u = (\mathsf{K}^u, \mathsf{sk}^u, \mathsf{vk}^u), w\right)$ generates $c \leftarrow \mathsf{Enc}\left(\mathsf{K}^u, w\right)$ and $\sigma \leftarrow \mathsf{Sign}\left(\mathsf{sk}^u, c\right)$, and outputs $(c, \sigma)$.
- $\mathsf{Search}\left(\Delta, q, T_S\right)$ outputs $\Delta\left(T_S, q\right)$.

The following claim states that if the obfuscator $\mathcal{O}$ used in Construction 2 is a secure diO obfuscator, and all other building blocks are secure, then Construction 2 is an MKSE scheme (as in Definition 3.1).

**Claim 5.4** (MKSE (Short Share Keys))**.** *Assume that Construction 2 is instantiated with:*

- *a secure diO obfuscator $\mathcal{O}$ for TMs,*
- *a family of collision-resistent hash functions $h$,*

$P_{\mathsf{K},(\mathsf{K}^u,\mathsf{vk}^u),h(T_S)}$

Hard-wired values: encryption keys $\mathsf{K}, \mathsf{K}^u$ (for a set and a querier, respectively), a signature verification key $\mathsf{vk}^u$, and a hash $h(T_S)$ of an (encrypted) set $T_S$.
Inputs: an (encrypted) set $T'_S$, a ciphertext $c$, and a signature $\sigma$.
Output: 0 (indicating that $c$ does not encrypt an element in the set encrypted by $T'_S$), or 1.
Operation:

1. Checks that $\mathsf{Ver}(\mathsf{vk}^v, \sigma, c) = 1$, otherwise outputs 0.
2. Checks that $h(T'_S) = h(T_S)$, otherwise outputs 0.
3. Initializes $\mathsf{found} = \mathsf{false}$.
4. Decrypts $w = \mathsf{Dec}(\mathsf{K}^u, c)$.
5. For every $c' \in T'_S$:
   (a) Decrypts $w' = \mathsf{Dec}(\mathsf{K}, c')$.
   (b) If $w = w'$ sets $\mathsf{found} = \mathsf{true}$.
6. Outputs $\mathsf{found}$.

Figure 1: Program $P_{\mathsf{K},(\mathsf{K}^u,\mathsf{vk}^u),h(T_S)}$ used to generate share keys in Construction 2

- *a double symmetric encryption scheme* $\left(\mathsf{KeyGen}^2, \mathsf{Enc}^2, \mathsf{Dec}^2\right)$, *and*
- *an existentially-unforgeable signature scheme,*

*then Construction 2 is a secure MKSE.*

*Moreover, if the encryption and signature schemes have* $\mathsf{poly}(\lambda)$*-length keys, and incur a* $\mathsf{poly}(\lambda)$ *overhead, then data and query keys, as well as queries, have length* $\mathsf{poly}(\lambda)$*, and for a set* $S$*, its corresponding processed set has size* $|S| \cdot \mathsf{poly}(\lambda)$*. Furthermore, if: (1) evaluating* $h$ *on length-*$n$ *inputs takes* $H_T(n)$ *time, and outputs a hash of length* $H_\ell(n)$*; and (2) there exist functions* $\mathsf{s}, T_{\mathcal{O}} : \mathbb{N} \to \mathbb{N}$ *such that for every TM* $M$*,* $|\mathcal{O}(M)| \leq \mathsf{s}(|M|)$*, and running* $\mathcal{O}(M)$ *on inputs of length* $n$ *takes* $T_{\mathcal{O}}(\mathsf{TIME}(M,n))$ *time, where* $\mathsf{TIME}(M,n)$ *is the running time of* $M$ *on length-*$n$ *inputs; then running* $\mathsf{Search}$ *on a set* $S$ *takes* $T_{\mathcal{O}}(H_T(|S| \cdot \mathsf{poly}(\lambda)) + |S| \cdot \mathsf{poly}(\lambda))$ *time, and share keys have size* $\mathsf{s}(H_\ell(|S| \cdot \mathsf{poly}(\lambda)) \cdot \mathsf{poly}(\lambda))$.

**Remark 5.5.** We note that the security of Construction 2 does not require the diO obfuscator to be secure with relation to *arbitrary* auxiliary inputs, but rather it is only required to guarantee security against a specific class of auxiliary inputs, as specified in the proof of Claim 5.4.

*Proof of Claim 5.4.* The correctness of the scheme follows directly from the correctness of the underlying primitives. We now argue that the scheme is secure.

Let $\mathcal{A}$ be a PPT adversary in the security game of Definition 3.1, let $\mathsf{K}_i$ be the data key $\mathcal{A}$ chose for data owner $i$, and let $\mathcal{W}^0, \mathcal{W}^1$ be the sets of queries $\mathcal{A}$ chose (for all other values chosen by $\mathcal{A}$, we use the notation of Definition 3.1). We proceed through a sequence of hybrids. Recall that the view of $\mathcal{A}$ in the security game consists of the encrypted sets $T_{S_i^b}$ for every $i \in \mathcal{D}$, queries $q$ for every $w \in \mathcal{W}^b$, and for every edge $(j,i) \in E$, the obfuscated program $\Delta_{j,i}$. In particular, since the keys, and encrypted sets, of corrupted data owners are identically distributed when $b = 0, 1$ (because $S_i^0 = S_i^1$ for every $i \in \mathcal{D}_c$, and they are encrypted using the same keys), we can fix these values into all hybrid

distributions, without decreasing the computational distance. Moreover, we assume without loss of generality that all data keys $\mathsf{K}_i$ chosen by $\mathcal{A}$ are valid keys. We now define the hybrids.

$\mathsf{view_0}$ : $\mathsf{view}_0$ is the view of $\mathcal{A}$ in the security game with $b = 0$.

$\mathcal{H}_0$ : In hybrid $\mathcal{H}_0$, the keys $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R), \mathsf{K}^u = (\mathsf{K}^u_L, \mathsf{K}^u_R)$ in every obfuscated program $P_{\mathsf{K},(\mathsf{K}^u,\mathsf{vk}^u),h\left(T_{S^0}\right)}$ are replaced with the keys $\mathsf{K}' = \left(\mathsf{K}_L, \vec{0}\right), \mathsf{K}^{u\prime} = \left(\mathsf{K}^u_L, \vec{0}\right)$.

$\mathcal{H}_0 \approx \mathsf{view}_0$ by the diO security of $\mathcal{O}$ (and a standard hybrid argument (over all obfuscated programs in $\mathsf{view}_0, \mathcal{H}_0$), because the TMs in each of the share keys $\Delta_{j,i}$ in $\mathsf{view}_0, \mathcal{H}_0$ are differing-inputs. Indeed, they are actually functionally equivalent (given *any* auxiliary input), since $\mathsf{Dec}^2$ completely disregards the right ciphertext, and so replacing the right secret key with the all-0 string does not affect functionality.

$\mathcal{H}_1$ : In hybrid $\mathcal{H}_1$, the encrypted set $T_i$ of every honest data owner $i \notin \mathcal{D}_c$ is generated as the encryption of $\left(S^0_i, S^1_i\right)$ with $\mathsf{Enc}^L$ (see Definition 5.7 below) instead of $\mathsf{Enc}^2$. (Notice that this also affects the share keys.)

To prove that $\mathcal{H}_0 \approx \mathcal{H}_1$, we will use the following lemma.

**Lemma 5.6.** *Let $\star \in \{L, R\}$. For every pair $(m_L, m_R)$ of messages, the following distributions are computationally indistinguishable, when $\mathsf{E}^2, \mathsf{E}^\star$ use the same underlying encryption scheme $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.*

- *$\mathcal{D}_1$: generate $K = (K_L, K_R) \leftarrow \mathsf{KeyGen}^2\left(1^\lambda\right)$ and $c \leftarrow \mathsf{Enc}^2\left(K, m_\star\right)$, and output $(K_\star, c)$.*
- *$\mathcal{D}_2$: generate $K = (K_L, K_R) \leftarrow \mathsf{KeyGen}^\star\left(1^\lambda\right)$ and $c \leftarrow \mathsf{Enc}^\star\left(K, (m_L, m_R)\right)$, and output $(K_\star, c)$.*

*Proof.* We prove the lemma for the case $\star = L$ (the case $\star = R$ is similar) by showing that indistinguishability follows from the security of the underlying scheme $\mathsf{E}$. Given a distinguisher $\mathsf{D}$ between $\mathcal{D}_1, \mathcal{D}_2$, we construct a distinguisher $\mathsf{D}'$ (that has $m_L$ hard-wired into it) between encryptions according to $\mathsf{E}$ of $m_L, m_R$. Given a ciphertext $c$, $\mathsf{D}'$ operates as follows: generates $\mathsf{K}' \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$, computes $c' \leftarrow \mathsf{Enc}\left(\mathsf{K}', m_L\right)$, and outputs $\mathsf{D}\left(\mathsf{K}', (c', c)\right)$. Notice that if $c$ encrypts $m_L$ then the input to $\mathsf{D}$ is distributed according to $\mathcal{D}_1$, otherwise it is distributed according to $\mathcal{D}_2$, so the distinguishing advantage of $\mathsf{D}'$ is equal to that of $\mathsf{D}$ which (by the security of $\mathsf{E}$) is negligible. □

By a standard hybrid argument, Lemma 5.6 implies that polynomially many ciphertexts (generated either by $\mathsf{E}^2$ or by $\mathsf{E}^L$, with the same or different keys), together with the keys of the "left" component, are computationally indistinguishable.

We prove $\mathcal{H}_0 \approx \mathcal{H}_1$ by reducing any distinguisher $\mathsf{D}$ between $\mathcal{H}_0, \mathcal{H}_1$ to a distinguisher $\mathsf{D}'$ between encryptions generated according to $\mathsf{E}^L$ or $\mathsf{E}^2$. We hard-wire into $\mathsf{D}'$ the querier keys and their queries, as well as the keys and encrypted sets of corrupted data owners, and the share keys associated with them. (This is possible because the encryption and signing keys of queriers, and their queries, are identically distributed in $\mathcal{H}_0, \mathcal{H}_1$, and independent of the encrypted sets; and since the share keys $\Delta_{j,i}$ for $i \in \mathcal{D}_c$ depend only on the keys of data owner $i$, querier $j$, and the encrypted set $T_i$, which are identically distributed in both hybrids.) $\mathsf{D}'$ operates as follows: given a sequence of ciphertexts (the encrypted sets of honest data owners), and the keys corresponding to the ciphertexts in the left components, $\mathsf{D}'$ honestly generates the hashes of encrypted sets of honest data owners, and uses the hard-wired querier keys, together with the keys for the left component in the ciphertexts of honest data owners, to generate the share keys between queriers and honest data owners. (Notice that since we have removed the key of the right component in ciphertexts of honest data owners, these are not needed to generate the share keys.) The values obtained in this way are distributed identically to $\mathcal{H}_0$

(if the input ciphertexts were generated with $\mathsf{E}^2$) or $\mathcal{H}_1$ (if they were generated with $\mathsf{E}^L$), so $\mathsf{D}'$ has the same distinguishing advantage as $\mathsf{D}$.

We now define the next hybrids.

$\mathcal{H}_2$ : In hybrid $\mathcal{H}_2$ the queries $\left(w^0_{j,1}, \cdots, w^0_{j,k_j}\right)$ of every querier $j \in \mathcal{Q}$ are generated using $\mathsf{Enc}^L$ with message $\left(w^0_{j,l}, w^1_{j,l}\right), 1 \leq l \leq k_j$, instead of $\mathsf{Enc}^2$.

$\mathcal{H}_1 \approx \mathcal{H}_2$ by a similar argument to the one used to show $\mathcal{H}_0 \approx \mathcal{H}_1$.

$\mathcal{H}_3$ : In hybrid $\mathcal{H}_3$, the generation of share keys $\Delta_{j,i}$ is modified as follows: (1) the hard-wired keys are $\left(\vec{0}, \mathsf{K}_{R,i}\right), \left(\vec{0}, \mathsf{K}^u_{R,j}\right)$, where $\left(\mathsf{K}_{L,i}, \mathsf{K}_{R,i}\right), \left(\mathsf{K}^u_{L,i}, \mathsf{K}^u_{R,j}\right)$ are the encryption keys of data owner $i$ and querier $j$, respectively; and (2) the program $P$ uses $\mathsf{Dec}^R$ (instead of $\mathsf{Dec}^2$) to decrypt $c$, and the elements of $T'_S$.

$\mathcal{H}_3 \approx \mathcal{H}_2$ by the diO security of $\mathcal{O}$, as we now show. Let $d$ denote the number of share keys available to the adversary (i.e., $d = |E|$), and order them in some arbitrary way: $\Delta^1, \cdots, \Delta^d$. We define a sequence of hybrids $\mathcal{H}^0, \cdots, \mathcal{H}^d$, where in $\mathcal{H}^l$, the first $l$ share keys are generated as in $\mathcal{H}_3$ (we denote these keys by $\Delta'_k$), and all other share keys are generated as in $\mathcal{H}_2$. We show that $\mathcal{H}^l \approx \mathcal{H}^{l-1}$ for every $1 \leq l \leq d$, and conclude that $\mathcal{H}_2 = \mathcal{H}^0 \approx \mathcal{H}^d = \mathcal{H}_3$.

Fix some $1 \leq l^* \leq d$, and let $(j,i)$ be the edge for which $\Delta^{l^*}$ was generated. We fix all the keywords $\mathcal{W}^0, \mathcal{W}^1$, and the sets $S^0_{i'}, S^1_{i'}$ for $i' \in \mathcal{D}$, into $\mathcal{H}^{l^*}, \mathcal{H}^{l^*-1}$ (this is possible because these values are identical in both hybrids). We additionally fix the keys of every querier $j', j' \neq j$ and data owner $i', i' \neq i$, the queries that querier $j'$ makes, the processed sets $T_{i'}, i' \neq i$, and share keys $\Delta_{j',i'}, \Delta'_{j',i'}$ (this is possible because these values are identically distributed in both hybrids). We now argue that $\Delta'_{j,i}, \Delta_{j,i}$ sampled in $\mathcal{H}^{l^*}, \mathcal{H}^{l^*-1}$ (respectively) form a differing-sinputs family of TMs with respect to the auxiliary information $\mathsf{aux}$ available to $\mathcal{A}$ (and so $\mathcal{H}^{l^*} \approx \mathcal{H}^{l^*-1}$ by the diO security of $\mathcal{O}$). This auxiliary information consists of the values we have fixed into $\mathcal{H}^{l^*}, \mathcal{H}^{l^*-1}$, the public verification key $\mathsf{vk}^u_j$ for signatures of querier $j$, all queries querier $j$ makes, the encrypted set $T_i$ of data owner $i$, and all $\Delta_{j,i'}, \Delta_{j',i}$ for $i' \neq i, j' \neq j$ such that $(j,i'), (j',i) \in E$. Let $\mathcal{P}, \mathcal{P}'$ denote the distributions over programs obfuscated in $\Delta_{j,i}, \Delta'_{j,i}$, and let $\mathsf{D}$ be a PPT algorithm that obtains $(P, P') \leftarrow \mathcal{P} \times \mathcal{P}'$ and $\mathsf{aux}$. In particular, notice that $\mathsf{D}$ knows the hash $h(T_i)$, and the encryption keys $\mathsf{K}_i, \mathsf{K}^u_j$ (but not the secret signing key $\mathsf{sk}^u_j$), since these appear in either $P$ or $P'$. We show that $\mathsf{D}$ succeeds in finding a differing input only with negligible probability.

Consider first the inputs (for $P, P'$) available to $\mathsf{D}$ in $\mathsf{aux}$, i.e., the encrypted set $T_i$ (which encrypts the elements of $S^0_i$ in the left components, and the elements of $S^1_i$ in the right components; this holds even if $i \in \mathcal{D}_c$ since in that case $S^0_i = S^1_i$), and the queries of querier $j$. For every such query $q$ there exists a pair $(w_L, w_R)$ of keywords such that $q$ is of the form $q = (c = (c_L, c_R), \sigma)$ where $c_\star \leftarrow \mathsf{Enc}\left(\mathsf{K}^u_{\star,j}, m_\star\right), \star \in \{L,R\}$, $\sigma \leftarrow \mathsf{Sign}\left(\mathsf{sk}^u_j, c\right)$, and $w_L \in S^0_i \Leftrightarrow w_R \in S^1_i$. (Here, $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is the encryption scheme used as a building block in the double encryption scheme.) In particular, $P(q, T_i) = P'(q, T_i)$ since the checks in Steps (1)-(2) succeed in both cases, and Steps (4)-(5) return the same outcome ($P$ searches for $w_L$ in $S^0_i$, since it decrypts using $\mathsf{Dec}^2$, whereas $P'$ searches for $w_R$ in $S^1_i$, since it decrypts with $\mathsf{Dec}^R$ and the right component of $T_i$ encrypts $S^1_i$; and $w_L \in S^0_i \Leftrightarrow w_R \in S^1_i$). In particular, both programs have the same running time in this case.

Next, we claim that for every other possible input $(T', c', \sigma')$ that $\mathsf{D}$ chooses (and does not appear in $\mathsf{aux}$), $P(T', c', \sigma') = P'(T', c', \sigma') = 0$ except with negligible probability. Indeed, if $(c', \sigma') \neq q$ (where $q$ is some query of querier $j$) then by the existential unforgeability of the signature scheme, with overwhelming probability $\sigma'$ is not a valid signature for $c'$, so the check in Step (1) fails in both $P, P'$ (in particular, both programs have the same running time in this case). Moreover, if $T' \neq T_i$

then by the collision resistance of $h$, with overwhelming probability $h(T') \neq h(T_i)$, so the check in Step (2) fails (and again, $P, P'$ have the same running time). Therefore, $P, P'$ are differing inputs with relation to the auxiliary information $\mathsf{aux}$.

We now define the final set of hybrids.

$\mathcal{H}_4$ : In hybrid $\mathcal{H}_4$, the queries $\left(w^1_{j,1}, \cdots, w^1_{j,k_j}\right)$ of every querier $j \in \mathcal{Q}$ are generated using $\mathsf{Enc}^2$ with message $\left(w^1_{j,l}, w^1_{j,l}\right), 1 \leq l \leq k_j$, instead of $\mathsf{Enc}^L$ with message $\left(w^0_{j,l}, w^1_{j,l}\right), 1 \leq l \leq k_j$.

$\mathcal{H}_3 \approx \mathcal{H}_4$ by a similar argument to the one used to prove $\mathcal{H}_1 \approx \mathcal{H}_2$.

$\mathcal{H}_5$ : In hybrid $\mathcal{H}_5$, the encrypted set $T_i$ of every honest data owner $i \notin \mathcal{D}_c$ is generated as the encryption of $\left(S^1_i, S^1_i\right)$ with $\mathsf{Enc}^2$ instead of with $\mathsf{Enc}^L$.

$\mathcal{H}_4 \approx \mathcal{H}_5$ by a similar argument to the one used to prove $\mathcal{H}_0 \approx \mathcal{H}_1$.

$\mathcal{H}_6$ : In hybrid $\mathcal{H}_6$, the obfuscated program for every edge $(j,i) \in E$ is generated as follows: (1) the hard-wired keys are $\mathsf{K} = \left(\mathsf{K}_{L,i}, \mathsf{K}_{R,i}\right), \mathsf{K} = \left(\mathsf{K}^u_{L,j}, \mathsf{K}^u_{R,j}\right)$, instead of $\mathsf{K}' = \left(\vec{0}, \mathsf{K}_{R,i}\right), \mathsf{K}^{u\prime} = \left(\vec{0}, \mathsf{K}^u_{R,j}\right)$; and (2) $\mathsf{Dec}^2$ is used for decryption (instead of $\mathsf{Dec}^R$).

$\mathcal{H}_6 \approx \mathcal{H}_5$ by the diO security of $\mathcal{O}$, using a standard hybrid argument in which the obfuscated programs are replaced one at a time. When replacing the program for edge $(j,i)$ from $P'$ (in $\mathcal{H}_5$) to $P$ (in $\mathcal{H}_6$), the PPT $\mathsf{D}$ (which should find a differing input) is given (as part of the auxiliary information $\mathsf{aux}$) the sets $S^1_{i'}, i' \in \mathcal{D}$; the keywords in $\mathcal{W}^1$ and the corresponding queries; the encryption keys of all data owners $i', i' \neq i$ and querier $j', j' \neq j$; the signing and verification keys of all queriers $j', j' \neq j$; and all encrypted sets $T_{i'}, i' \in \mathcal{D}$. Also, from $P, P'$ the distinguisher learns the encryption keys $\mathsf{K}_i, \mathsf{K}^u_j$, and the verification key $\mathsf{vk}^u_j$. We show that except with negligible probability, $\mathsf{D}$ fails to find a differing input (the argument is similar to that used to prove $\mathcal{H}_3 \approx \mathcal{H}_2$).

The inputs (for $P, P'$) available to $\mathsf{D}$ in $\mathsf{aux}$, i.e., the encrypted set $T_i$, and queries $q = (c, \sigma)$ of querier $j$ (where $c \leftarrow \mathsf{Enc}^2\left(\mathsf{K}^u_j, m\right)$ for some $m$, and $\sigma \leftarrow \mathsf{Sign}\left(\mathsf{sk}^u_j, c\right)$), are not differing-inputs (even though $P'$ decrypts the right component of $c$, whereas $P$ decrypts the left component) because both components of $c$ encrypt $m$ according to $\mathsf{Enc}$ (so both $P, P'$ search for $m$ in $S^1_i$). For every other possible input $(T', c', \sigma')$ that $\mathsf{D}$ chooses, $P(T', c', \sigma') = P'(T', c', \sigma') = 0$ except with negligible probability: if $(c', \sigma') \neq q$ (where $q$ is some query of querier $j$) then with overwhelming probability $\sigma'$ is not a valid signature on $c'$ (by the existential unforgeability of the signature scheme); whereas if $T' \neq T_i$ then with overwhelming probability $h(T') \neq h(T_i)$ (by the collision resistance of $h$). Consequently, $\mathcal{H}_5 \approx \mathcal{H}_6$. Since $\mathcal{H}_6$ is the view of $\mathcal{A}$ in the security game with $b = 1$, we conclude that $\mathcal{A}$ has only negligible advantage in the security game.

Finally, we analyze the complexity of the scheme. Data and query keys, which are simply encryption and signing keys, have size $\mathsf{poly}(\lambda)$. Queries are ciphertext for length-$O(\lambda)$ keywords (since the universe is at most of size $2^\lambda$), together with signatures on these ciphertexts. Similarly, a processed set consists of encryptions of each of its keywords, so its size is $|S| \cdot \mathsf{poly}(\lambda)$. Regarding share keys, the TM has size $H_\ell(|S| \cdot \mathsf{poly}(\lambda)) \cdot \mathsf{poly}(\lambda)$ (since $|h(T_S)| \leq H_\ell(|S| \cdot \mathsf{poly}(\lambda))$), and so by the assumption on the blowup caused by obfuscation, share keys have size $\mathsf{s}(H_\ell(|S| \cdot \mathsf{poly}(\lambda)) \cdot \mathsf{poly}(\lambda))$. Finally, $\mathsf{Search}$ consists of running the obfuscated TM, which requires computing the hash ($H_T(|S| \cdot \mathsf{poly}(\lambda))$ time), and performing $O(|S|)$ operations, each taking $\mathsf{poly}(\lambda)$ time, so $\mathsf{TIME}(M, |S|) = H_T(|S| \cdot \mathsf{poly}(\lambda)) + |S| \cdot \mathsf{poly}(\lambda)$, and consequently the running time is $T_{\mathcal{O}}(H_T(|S| \cdot \mathsf{poly}(\lambda)) + |S| \cdot \mathsf{poly}(\lambda))$. □

The following encryption scheme was used to prove Claim 5.4:

**Definition 5.7.** Given a symmetric encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, and $\star \in \{L, R\}$, we define an encryption scheme $\mathsf{E}^\star = (\mathsf{KeyGen}^\star, \mathsf{Enc}^\star, \mathsf{Dec}^\star)$ as follows:

- $\mathsf{KeyGen}^\star$ operates as $\mathsf{KeyGen}^2$ from Definition 5.3. That is, on input $1^\lambda$ it generates $\mathsf{K}_L \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$ and $\mathsf{K}_R \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$, and outputs $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R)$.
- $\mathsf{Enc}^\star$, on input a key $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R)$ and a message $m = (m_L, m_R)$, computes $c_L \leftarrow \mathsf{Enc}\left(\mathsf{K}_L, m_L\right)$ and $c_R \leftarrow \mathsf{Enc}\left(\mathsf{K}_R, m_R\right)$, and outputs $c = (c_L, c_R)$.
- $\mathsf{Dec}^\star$, on input a key $\mathsf{K} = (\mathsf{K}_L, \mathsf{K}_R)$ and a ciphertext $c = (c_L, c_R)$, outputs $\mathsf{Dec}\left(\mathsf{K}_\star, c_\star\right)$.

The proof of Theorem 5.1 now follows as a corollary from Claim 5.4.

*Proof of Theorem 5.1.* We instantiate Construction 2 with the double encryption scheme of Definition 5.3, based on the encryption scheme whose existence follows from the existence of a CRHF; the hash function with a Merkle Hash Tree (MHT) hash based on the CRHF; and instantiate $\mathcal{O}$ with the diO obfuscator. Then the security of the scheme, as well as the length of data and query keys, queries, and processed sets, follow directly from Claim 5.4. Regarding share keys, the MHT has $\mathsf{poly}(\lambda)$-length outputs, and $\mathcal{O}$ causes only a polynomial blowup, so by Claim 5.4, share keys have length $\mathsf{poly}(\lambda)$. Finally, generating the MHT for a set of size $s$ takes time $s \cdot \mathsf{poly}(\lambda)$, and so the runtime of $\mathsf{Search}$ is $\mathsf{poly}(\lambda, |S|)$. □

## 5.2 MKSE from Public-Coin Differing-Inputs Obfuscation

In this section we show that a slight modification of Construction 2 is secure assuming the underlying obfuscator is a *pc*-diO obfuscator for TMs. More specifically, we only need to use a signature scheme with some "special" properties. Concretely, we prove the following for a universe $\mathcal{U}$ of size $|\mathcal{U}| \leq \mathsf{poly}\left(2^\lambda\right)$:

**Theorem 5.8** (MKSE from pc-diO (short share keys))**.** *Assume that OWPs, CRHFs, and pc-diO for TMs with polynomial blowup, exist. Then there exists a secure MKSE in which share keys have size* $\mathsf{poly}(\lambda)$*, where* $\lambda$ *is a security parameter. Moreover, data and query keys, as well as queries, have length* $\mathsf{poly}(\lambda)$*, and given a set* $S$*, its processed version has size* $|S| \cdot \mathsf{poly}(\lambda)$*, and searching in it takes* $\mathsf{poly}(\lambda, |S|)$ *time.*

The reason we need to change the MKSE scheme outlined in Section 5.1 is that it cannot use a *pc*-diO obfuscator. (Roughly speaking, pc-diO guarantees indistinguishability of the obfuscated programs only as long as it is infeasible for a PPT adversary to find an input on which they differ, even given the randomness used to sample the programs; see Definition A.4 in Appendix A.) Indeed, for every querier $j$ and data owner $i$, the randomness used to sample the program $P_{j,i}$ (i.e., the program obfuscated in the share key $\Delta_{j,i}$) includes the signing key of querier $j$. This allows one to sign arbitrary messages, meaning the obfuscated programs in the security game when $b = 0$ and $b = 1$ are *not* differing-inputs. (The program $P_{j,i}$ for querier $j$ and data owner $i$ contains $h\left(S_i^0\right)$ when $b = 0$, and $h\left(S_i^1\right)$ when $b = 1$, so a differing input would be a query on any keyword contained in one and not the other. The query can be efficiently generated since the encryption and signing keys appear in the randomness used to sample $P_{j,i}$.)

To overcome this issue, we introduce (Section 5.2.1) a new signature primitive which we call *dual-mode* signatures. Roughly, a dual-mode signature scheme is an existentially-unforgeable signature scheme associated with an additional $\mathsf{SpecialGen}$ algorithm that given a list of messages, generates "fake" signatures on these messages, and a "fake" verification key under which they can be verified. These "fake" signatures and key are computationally indistinguishable from honestly generated signatures and verification key, and the "fake" verification key cannot be used to successfully sign other

messages, even given the randomness used to generate the "special mode" verification key and signatures. (This rules out the trivial construction in which $\mathsf{SpecialGen}$ simply runs the key generation and signing algorithms.)

One can think of the $\mathsf{SpecialGen}$ algorithm as a way of "puncturing" the signing key from the procedure that generates the verification key and signatures. We use this viewpoint to prove (Section 5.2.2) security based on pc-diO security and dual-mode signatures: we first replace the actual verification key used in $P_{j,i}$, and the signatures in the queries of $j$, with ones generated by $\mathsf{SpecialGen}$; and then use pc-diO security to replace the obfuscated program from one containing $h\left(S_i^0\right)$ to one containing $h\left(S_i^1\right)$. (Notice that now the randomness used to sample $P_{j,i}$ does *not* contain the signing key, so one cannot sign queries that $j$ did not issue.)

### 5.2.1 Dual-Mode Signatures

We now formally define the notion of dual-mode signatures, and construct such signatures based on OWPs and CRHFs.

**Definition 5.9** (Dual-mode signature scheme)**.** We say that an existentially-unforgeable signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ is a *dual-mode signature scheme* if there exists a PPT algorithm $\mathsf{SpecialGen}$ such that the following holds.

- **Special mode generation.** $\mathsf{SpecialGen}$, on input a security parameter $1^\lambda$, an input length $1^n$, and a non-empty set $\left\{m_1, \cdots, m_{k(\lambda)}\right\}$ of length-$n$ messages (for some $k(\lambda) = \mathsf{poly}(\lambda)$), outputs a verification key $\mathsf{vk}$, and a set of signatures $\left\{\sigma_1, \cdots, \sigma_{k(\lambda)}\right\}$.
- **Standard and special modes are indistinguishable.** For every $k(\lambda) = \mathsf{poly}(\lambda)$, every input length $n \in \mathbb{N}$, and every non-empty set $\left\{m_1, \cdots, m_{k(\lambda)}\right\}$ of length-$n$ messages, the following distributions are computationally indistinguishable:
  - $\mathcal{D}_{\text{standard}}\left(1^\lambda\right)$: generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}\left(1^\lambda, 1^n\right)$; for every $1 \leq i \leq k(\lambda)$, compute $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$; output $\left(\mathsf{vk}, \left\{\sigma_1, \cdots, \sigma_{k(\lambda)}\right\}\right)$.
  - $\mathcal{D}_{\text{special}}\left(1^\lambda\right)$: run $\left(\mathsf{vk}, \left\{\sigma_1, \cdots, \sigma_{k(\lambda)}\right\}\right) \leftarrow \mathsf{SpecialGen}\left(1^\lambda, 1^n, \left\{m_1, \cdots, m_{k(\lambda)}\right\}\right)$; output $\left(\mathsf{vk}, \left\{\sigma_1, \cdots, \sigma_{k(\lambda)}\right\}\right)$.
- **Special mode cannot sign additional messages.** For every input length $n = \mathsf{poly}(\lambda)$, and polynomial-time forger $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following probability is negligible in $\lambda$:

$$\Pr\left[m \notin M \wedge \mathsf{Ver}(\mathsf{vk}, \sigma, m) = 1 : \begin{array}{c}\left(M = \left\{m_1, \cdots, m_{|M|}\right\}, \mathsf{St}\right) \leftarrow \mathcal{A}_1\left(1^\lambda\right) \\ \left(\mathsf{vk}, \left\{\sigma_1, \cdots, \sigma_{|M|}\right\}\right) \leftarrow \mathsf{SpecialGen}\left(1^\lambda, 1^n, M; r\right) \\ (\sigma, m) \leftarrow \mathcal{A}_2\left(\mathsf{St}, \left(\mathsf{vk}, \left\{\sigma_1, \cdots, \sigma_{|M|}\right\}\right), r\right)\end{array}\right]$$

Notice that $\mathcal{A}_2$ cannot forge signatures, *even given the randomness used to generate the "special mode" verification key and signatures.* This rules out the trivial construction in which $\mathsf{SpecialGen}$ simply emulates $\mathcal{D}_{\text{standard}}$.

**Remark 5.10.** The indistinguishability of standard and special modes property guarantees that special-mode verification keys are indistinguishable from standard-mode verification keys (regardless of the messages for which the special-mode verification key was generated).

We construct dual-mode signatures from OWPs and CRHFs in two steps. We first construct a *one-time* dual-mode signature scheme, in which $\mathsf{SpecialGen}$ works only for a *single* message, and security holds only for $\mathcal{A}_1$ that chooses a *singleton* $M$. These one-time signatures are based on Lamport's one-time signatures [Lam]. Then, we show a general transformation from one-time to multi-time dual-mode signatures.

**Construction 3** (One-time dual-mode signatures). The scheme uses a family $\mathcal{P} = \left\{P_\lambda : \{0,1\}^\lambda \to \{0,1\}^\lambda\right\}$ of OWPs, and a hash function $h : \{0,1\}^* \to \{0,1\}^\kappa$ (for some $\kappa \in \mathbb{N}$), and consists of the following algorithms:

- $\mathsf{KeyGen}$ on input a security parameter $1^\lambda$, and an input length $1^n$, picks $x_i^b \in_R \{0,1\}^\lambda$ for every $b \in \{0,1\}, 1 \leq i \leq \kappa$, computes $y_i^b = P_\lambda\left(x_i^b\right)$, and outputs

$$\left(\mathsf{sk} = \left\{x_i^b\right\}_{b\in\{0,1\},1\leq i\leq\kappa}, \mathsf{vk} = \left\{y_i^b\right\}_{b\in\{0,1\},1\leq i\leq\kappa}\right).$$

- $\mathsf{Sign}$ on input a signing key $\mathsf{sk} = \left\{x_i^b\right\}_{b\in\{0,1\},1\leq i\leq\kappa}$, and a message $m = m_1 \cdots m_n \in \{0,1\}^n$, computes $z = h(m)$, and outputs $\sigma = \left(x_i^{z_i}\right)_{1\leq i\leq\kappa}$.

- $\mathsf{Ver}$ on input a verification key $\mathsf{vk} = \left\{y_i^b\right\}_{b\in\{0,1\},1\leq i\leq\kappa}$, a signature $\sigma = \left(x_i'\right)_{1\leq i\leq\kappa}$, and a message $m = m_1 \cdots m_n \in \{0,1\}^n$, computes $z = h(m)$, and outputs 1 if and only if $P_\lambda\left(x_i'\right) = y_i^{z_i}$ for every $1 \leq i \leq \kappa$.

- $\mathsf{SpecialGen}$ on input a security parameter $1^\lambda$, an input length $1^n$, and a message $m = m_1 \cdots m_n \in \{0,1\}^n$, operates as follows: computes $z = h(m)$, then for every $1 \leq i \leq \kappa$, picks $x_i^{z_i} \in_R \{0,1\}^\lambda$, computes $y_i^{z_i} = P_\lambda\left(x_i^b\right)$, and picks $y_i^{\bar{z}_i} \in_R \{0,1\}^\lambda$; and outputs $\left(\mathsf{vk} = \left\{y_i^b\right\}_{b\in\{0,1\},1\leq i\leq\kappa}, \sigma = \left(x_i^{z_i}\right)_{1\leq i\leq\kappa}\right)$.

We note that our construction uses a *OWP*, whereas standard Lamport signatures can use any *OWF*. A permutation is needed to guarantee that the special and standard modes are indistinguishable. Next, we prove that Construction 3 is a one-time dual-mode signature scheme.

**Lemma 5.11.** *Assume OWPs and CRHFs exist. Then there exists a one-time dual-mode signature scheme. Moreover, there exists a polynomial $p(\lambda)$ such that signatures have length $p(\lambda)$, and signing and verification keys have length $2p(\lambda)$ (for any input length $n = \mathsf{poly}(\lambda)$).*

*Proof.* We show that Construction 3 has the appropriate properties. To obtain $\mathsf{poly}(\lambda)$-length keys, we can instantiate the construction with any family of polynomially-secure CRHFs, and obtain keys of length $\mathsf{poly}(\lambda)$ using a MHT (if hash functions in the family are not sufficiently compressing). We now discuss the properties of the scheme.

**Existential unforgeability.** $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ is Lamport's one-time signature scheme [Lam] and so it is existentially unforgeable.

**Standard and special modes are indistinguishable.** The only difference between $\mathcal{D}_{\text{standard}}$ and $\mathcal{D}_{\text{special}}$ for a message $m \in \{0,1\}^n$ is in the values $y_i^{\bar{z}_i}$, for $z = h(m)$, that appear in the verification key (all other values are identically distributed in both distributions): in $\mathcal{D}_{\text{standard}}$ these are the images (under $P_\lambda$) of random values, whereas in $\mathcal{D}_{\text{special}}$ these are random values. But since $P_\lambda$ is a permutation, the images of random values are themselves random, so $\mathcal{D}_{\text{standard}} \approx \mathcal{D}_{\text{special}}$.

**Special mode cannot sign additional messages.** This follows from the security of the OWP. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-time forger in Definition 5.9 such that

$$\Pr\left[m' \neq m \wedge \mathsf{Ver}\left(\mathsf{vk}, \sigma', m'\right) = 1 \;:\; \begin{array}{c}(m, \mathsf{St}) \leftarrow \mathcal{A}_1\left(1^\lambda\right) \\ (\mathsf{vk}, \sigma) \leftarrow \mathsf{SpecialGen}\left(1^\lambda, 1^n, \{m\}; r\right) \\ (\sigma', m') \leftarrow \mathcal{A}_2\left(\mathsf{St}, (\mathsf{vk}, \sigma), r\right)\end{array}\right] = \epsilon(\lambda).$$

We construct an adversary $\mathcal{A}'$ that succeeds in inverting $P_\lambda$ with probability $\frac{\epsilon(\lambda)}{\kappa} \cdot (1 - \mathsf{negl}(\lambda))$, and conclude from the one-wayness of $P_\lambda$ that $\epsilon(\lambda) = \mathsf{negl}(\lambda)$ (since $\kappa = \mathsf{poly}(\lambda)$). $\mathcal{A}'$ operates as follows:

it runs $\mathcal{A}_1$ to obtain a message $m \in \{0,1\}^n$ to sign, and a state $\mathsf{St}$. $\mathcal{A}$ then chooses $i^* \in_R [\kappa]$. Given a challenge $y$ (the image under $P_\lambda$ of a random $x$), $\mathcal{A}$ generates the verification key $\mathsf{vk}$, and signature $\sigma$ on $m$, by emulating $\mathsf{SpecialGen}$ with the following modification: it sets $y_{i^*}^{\bar{z}_{i^*}} = y$ (instead of picking it at random), where $z = h(m)$. Let $r$ denote the random bits which $\mathcal{A}'$ used to emulate $\mathsf{SpecialGen}$ (i.e., the bits used to sample the $x_i^{z_i}$'s and the $y_i^{\bar{z}_i}$'s), together with the value $y$ (which are the random bits that would give $y_{i^*}^{\bar{z}_{i^*}}$ the value $y$). Then $\mathcal{A}'$ runs $\mathcal{A}_2$ with input $\left(\mathsf{St}, \left(\mathsf{vk}, \{x_i^{z_i}\}_{1 \le i \le \kappa}\right), r\right)$, and obtains a message $m'$, and a purported signature $\sigma' = \{x_i'\}_{1 \le i \le \kappa}$. $\mathcal{A}'$ outputs $x'_{i^*}$ as the pre-image of $y$. We claim that $\Pr\left[P_\lambda\left(x'_{i^*}\right) = y\right] \ge \frac{\epsilon(\lambda)}{\kappa} \cdot (1 - \mathsf{negl}(\lambda))$. Notice first that the input to $\mathcal{A}_2$ is identically distributed to its view in the forging game. Indeed, the only difference is in the choice of $y_{i^*}^{\bar{z}_{i^*}}$: in the forging game it is chosen uniformly at random, whereas $\mathcal{A}'$ sets it to be the image of a random value, but since $P_\lambda$ is a permutation, it is also randomly distributed. Moreover, if $m \neq m'$ then except with negligible probability, $h(m) \neq h(m')$, from the collision-resistance of $h$. Therefore,

$$\Pr\left[P_\lambda\left(x'_{i^*}\right) = y\right] = \Pr\left[\mathsf{Ver}\left(\mathsf{vk}, \sigma', m'\right) = 1 \wedge m' \neq m\right] \cdot \Pr\left[h(m)_{i^*} \neq h\left(m'\right)_{i^*} | m \neq m'\right]$$

$$\geq \epsilon(\lambda) \cdot \frac{1}{\kappa}\left(1 - \mathsf{negl}(\lambda)\right)$$

(where the rightmost inequality holds because $i^*$ is random and unknown to $\mathcal{A}_2$).

□

We now describe a transformation from one-time dual-mode signature schemes to (multi-time) dual-mode signature schemes which, combined with Construction 3, gives a construction of a (multi-use) dual-mode signature scheme. The high-level idea of the transformation is to use the hash-tree construction of Naor and Yung [NY89], extending it to also support special-mode generation.

**Construction 4** (Dual-mode signature scheme)**.** The dual-mode signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{SpecialGen})$ uses a one-time dual-mode signature scheme $\left(\mathsf{KeyGen}^1, \mathsf{Sign}^1, \mathsf{Ver}^1, \mathsf{SpecialGen}^1\right)$ where for every $n = \mathsf{poly}(\lambda)$, $\mathsf{KeyGen}^1\left(1^\lambda, 1^n\right)$ outputs verification keys of length $\kappa = \kappa(\lambda)$; and a PRF $F$. The scheme consists of the following algorithms.

- $\mathsf{KeyGen}$ on input a security parameter $1^\lambda$, and an input length $1^n$, generates a pair of one-time signing and verification keys $\left(\mathsf{sk}^1, \mathsf{vk}^1\right) \leftarrow \mathsf{KeyGen}^1\left(1^\lambda, 1^{2\kappa}\right)$, and a random key $K \in_R \{0,1\}^\lambda$ for $F$, and outputs $\left(\mathsf{sk} = \left(\mathsf{sk}^1, K\right), \mathsf{vk} = \mathsf{vk}^1\right)$.
- $\mathsf{Sign}$ on input a signing key $\mathsf{sk} = \left(\mathsf{sk}^1, K\right)$, and a message $m = m_1 \cdots m_n \in \{0,1\}^n$, operates as follows:
  - Interprets $m$ as a leaf in a depth-$n$ full binary tree in which level$-i$ nodes are associated with strings in $\{0,1\}^i$, where the left (right) child of a node associated with string $s$, is associated with the string $s0$ ($s1$). (In particular, the root, which is at level 0, is associated with the empty string $\epsilon$.)
  - Assigns signing and verification key pairs for internal nodes on the path from the root to $m$ (excluding the root, and the leaf $m$), and their siblings, as follows: for every such level-$i$ node associated with string $s$, assigns to the node the keys $(\mathsf{sk}_s, \mathsf{vk}_s)$ generated as: $(\mathsf{sk}_s, \mathsf{vk}_s) := \mathsf{KeyGen}^1\left(1^\lambda, 1^{2\kappa}; F_K\left(i \circ s0^{n-i}\right)\right)$ (recall that $\circ$ denotes string concatenation; $i$ here denotes the $\log n$-bit binary representation of $i$).
  - Assigns signing and verification key pairs for the leaf $m$, and its sibling $m' = m_1 \cdots m_{n-1}\bar{m}_n$, as $(\mathsf{sk}_m, \mathsf{vk}_m) := \mathsf{KeyGen}^1\left(1^\lambda, 1^n; F_K(n \circ m)\right)$, and $(\mathsf{sk}_{m'}, \mathsf{vk}_{m'}) := \mathsf{KeyGen}^1\left(1^\lambda, 1^n; F_K(n \circ m')\right)$, respectively. (Again, $n \circ m$ denotes the $\log n$-bit binary representation of $n$, concatenated with $m$.)

- Computes $\sigma_n \leftarrow \mathsf{Sign}^1(\mathsf{sk}_m, m)$. For every $0 \leq i < n$, let $m^i = m_1 \cdots m_i$, then $\mathsf{Sign}$ computes $\sigma_i \leftarrow \mathsf{Sign}^1(\mathsf{sk}_{m^i}, (\mathsf{vk}_{m^i 0}, \mathsf{vk}_{m^i 1}))$ (where $\mathsf{sk}_{m^0} = \mathsf{sk}_\epsilon = \mathsf{sk}^1$).
- Outputs $\sigma = \left((\sigma_0, \cdots, \sigma_n), \left(\mathsf{vk}_{m_1 \cdots m_{i-1} 0}, \mathsf{vk}_{m_1 \cdots m_{i-1} 1}\right)_{1 \leq i \leq n}\right)$.

- $\mathsf{Ver}$ on input a verification key $\mathsf{vk}$, a signature $\sigma = \left((\sigma_0, \cdots, \sigma_n), \left(\mathsf{vk}_i^0, \mathsf{vk}_i^1\right)_{1 \leq i \leq n}\right)$, and a message $m = m_1 \cdots m_n \in \{0,1\}^n$, outputs 1 if and only if all the following checks pass:
  - $\mathsf{Ver}^1(\mathsf{vk}_n^{m_n}, \sigma_n, m) = 1$.
  - For every $1 \leq i < n$, $\mathsf{Ver}^1\left(\mathsf{vk}_i^{m_i}, \sigma_i, \left(\mathsf{vk}_{i+1}^0, \mathsf{vk}_{i+1}^1\right)\right) = 1$.
  - $\mathsf{Ver}^1\left(\mathsf{vk}, \sigma_0, \left(\mathsf{vk}_1^0, \mathsf{vk}_1^1\right)\right) = 1$.

- $\mathsf{SpecialGen}$ on input a security parameter $1^\lambda$, an input length $1^n$, and a non-empty set of length-$n$ messages $M = \left\{m^1, \cdots, m^k\right\}$, operates as follows:
  - Generates a (non-full) binary tree in the following way:
    - For every $m \in M$, generates a level-$n$ verification key, and signature for $m$, as $(\mathsf{vk}_m, \sigma_m) \leftarrow \mathsf{SpecialGen}^1\left(1^\lambda, 1^n, \{m\}\right)$.
    - We say that a node is *non-empty* if some verification key has already been associated with it, otherwise we say the node is *empty*. For every level $i$ from $n-1$ to 0, and for every level-$i$ node $s$ (from left to right) that has a non-empty child:
      - Let $\mathsf{vk}_{s0}, \mathsf{vk}_{s1}$ be the verification keys associated with the children of $s$, where if its left (right) child is empty then $\mathsf{vk}_{s0} \leftarrow \mathsf{SpecialGen}^1\left(1^\lambda, 1^{2\kappa}, \{\vec{0}\}\right)$ ($\mathsf{vk}_{s1} \leftarrow \mathsf{SpecialGen}^1\left(1^\lambda, 1^{2\kappa}, \{\vec{0}\}\right)$).
      - Generate a verification key $\mathsf{vk}_s$ for $s$, and a signature for $(\mathsf{vk}_{s0}, \mathsf{vk}_{s1})$, as $(\mathsf{vk}_s, \sigma_s) \leftarrow \mathsf{SpecialGen}^1\left(1^\lambda, 1^{2\kappa}, \{(\mathsf{vk}_{s0}, \mathsf{vk}_{s1})\}\right)$.
  - For every $m \in M$, sets its signature to be $\sigma^m = \left((\sigma_{m^i})_{0 \leq i \leq n}, (\mathsf{vk}_{m^{i-1} 0}, \mathsf{vk}_{m^{i-1} 1})_{1 \leq i \leq n}\right)$, where $m^i = m_1 \cdots m_i$ for every $0 \leq i \leq n$.
  - Output $(\mathsf{vk}_\epsilon, \{\sigma^m : m \in M\})$.

**Lemma 5.12.** *If $\left(\mathsf{KeyGen}^1, \mathsf{Sign}^1, \mathsf{Ver}^1, \mathsf{SpecialGen}^1\right)$ is a one-time dual-mode signature scheme, then Construction 4 is a dual-mode signature scheme.*

*Moreover, assume that $\mathsf{KeyGen}^1$ on input $1^\lambda, 1^n$ outputs signing and verification keys of length $\kappa = \kappa(\lambda)$, and $\mathsf{Sign}^1$ on input $1^\lambda$, and a length-$n$ message, outputs signatures of length $\ell = \ell(\lambda)$. Then $\mathsf{KeyGen}$ on input $1^\lambda, 1^n$ outputs a signing key of length $\kappa + \lambda$ and a verification key of length $\kappa$, and $\mathsf{Sign}$ on input $1^\lambda$, and a length-$n$ message, outputs signatures of length $(n+1) \cdot (2\kappa + \ell)$.*

*Proof.* The claim regarding the complexity of the scheme follows directly from the construction, and the assumption regarding the complexity of $\mathsf{KeyGen}^1, \mathsf{Sign}^1$. We now discuss the properties of the construction.

**Existential unforgeability.** Let $\mathcal{A}$ be a polynomial time adversary that succeeds in forging a signature with probability $\epsilon(\lambda)$. We use it to construct an adversary $\mathcal{A}'$ that succeeds in forging a signature in the underlying one-time signature scheme with probability $\frac{\epsilon(\lambda)}{n \cdot p(\lambda)} - \mathsf{negl}(\lambda)$, where $p(\lambda) = \mathsf{poly}(\lambda)$ bounds the running time of $\mathcal{A}$ (and consequently, also the number of signatures it requests to receive), and conclude (from the existential-unforgeability of the one-time scheme, and since $n = \mathsf{poly}(\lambda)$) that $\epsilon(\lambda) = \mathsf{negl}(\lambda)$.

$\mathcal{A}'$ operates as follows. It obtains a verification key $\mathsf{vk}$ from its challenger, and honestly generates one-time signing and verification keys $\left(\mathsf{sk}^1, \mathsf{vk}^1\right)$ for the root of the tree, and the key $K$ for $F$. Then, $\mathcal{A}'$ picks $i^* \in_R [p(\lambda)]$ and $j^* \in_R \{0, \cdots, n-1\}$. (Intuitively, $\mathcal{A}'$ guesses that the following holds for the message $m'$ on which $\mathcal{A}$ will try to forge a signature: its path in the tree, and the path of the $i^*$'th message on which $\mathcal{A}$ will request a signature, have the longest intersection (i.e., agree on the longest prefix of a path, out of all messages on which $\mathcal{A}$ requests a signature; if there are several such messages, then by choosing $i^*$, $\mathcal{A}'$ guesses that $i^*$ is the first message for which this holds), and these paths coincide up to level $j^*$.) Then, $\mathcal{A}'$ emulates $\mathcal{A}$, where any signing request that $\mathcal{A}$ makes on message $m$ is answered as follows. If $m$ is not the $i^*$'th signing request of $\mathcal{A}$, then $\mathcal{A}'$ generates the signature on $m$ by honestly evaluating the $\mathsf{Sign}$ algorithm. If $m$ is the $i^*$'th signing request, then $\mathcal{A}'$ generates a signature on $m$ by honestly evaluating the $\mathsf{Sign}$ algorithm, except for the following modification: it uses $\mathsf{vk}$ as the verification key of the node $v$ which is the sibling of the level-$(j^*+1)$ node on the path to $m$ (instead of choosing it as specified by the $\mathsf{Sign}$ algorithm). We note that if $\mathcal{A}'$ was wrong in its guess for $i^*, j^*$, then $v$ might (have already, or will later) appear on the path to some message on which $\mathcal{A}$ requests a signature (if $v$ has already appeared, its verification key has already been assigned; if it will appear in a later request, then $\mathcal{A}'$ will not be able to generate a signature, since it does not know the corresponding signing key). If this happens, $\mathcal{A}'$ aborts the forging game. At some point, $\mathcal{A}$ provides a message $m'$, and a purported signature $\sigma' = \left(\left(\sigma_0, \cdots, \sigma_n\right), \left(\mathsf{vk}_i^0, \mathsf{vk}_i^1\right)_{1 \leq i \leq n}\right)$ on $m'$. Finally, $\mathcal{A}'$ sends $\sigma_{j^*+1}$ as the signature on $\left(\mathsf{vk}_{j^*+2}^0, \mathsf{vk}_{j^*+2}^1\right)$ (if $j^* < n-1$) or on $m'$ (if $j^* = n-1$).

To analyze the success probability of $\mathcal{A}'$, we consider a hybrid distribution $\mathcal{H}$ which consists of the level-$(j^*+1)$ message on the path to $m'$, and the signature $\sigma_{j^*+1}$, in the above interaction between $\mathcal{A}, \mathcal{A}'$, where the challenge verification key $\mathsf{vk}$ is generated by running $\mathsf{KeyGen}^1\left(1^\lambda, 1^t; F_K\left((j^*+1) \circ \widetilde{m}^{j^*} 0^{n-j^*-1}\right)\right)$, where $t = 2\kappa$ if $j^* < n-1$, otherwise $t = n$, and $\widetilde{m}^{j^*} = \widetilde{m}_1 \cdots \widetilde{m}_{j^*} \widetilde{m}_{j^*+1}$ where $\widetilde{m}$ is the $i^*$'th message on which $\mathcal{A}$ requests a signature. (That is, instead of running $\mathsf{KeyGen}^1$ with uniformly random bits, as is the case in the forging game of $\mathcal{A}'$, it is run with the same random bits with which $\mathsf{KeyGen}$ would evaluate it. Notice that this requires knowing $\widetilde{m}^{j^*}$ in advance, which is possible because $\mathcal{A}'$ can choose $i^*, j^*$, and emulate the signatures on the first $i^*-1$ messages, before obtaining $\mathsf{vk}$. Therefore, by the time $\mathsf{vk}$ is needed, $\widetilde{m}^{j^*}$ has already been determined.) Notice first that in $\mathcal{H}$, $\mathcal{A}'$ perfectly emulates the challenger of $\mathcal{A}$. Indeed, the only difference was in how $\mathcal{A}'$ chose the verification key for node $\widetilde{m}^{j^*}$, which in $\mathcal{H}$ is chosen exactly as the $\mathsf{Sign}$ algorithm chooses it. Therefore, if $\sigma'$ is a valid signature on $m'$ (which happens with probability $\epsilon(\lambda)$), then in particular $\mathsf{Ver}^1\left(\mathsf{vk}, \sigma_{j^*+1}, m''\right) = 1$ (where $m'' = \left(\mathsf{vk}_{j^*+2}^0, \mathsf{vk}_{j^*+2}^1\right)$ if $j^* < n-1$, otherwise $m'' = m'$), meaning that if $\mathcal{A}'$ guessed $i^*, j^*$ correctly (which happens with probability $\frac{1}{n \cdot p(\lambda)}$), then $\mathcal{A}'$ succeeds in forging a signature in $\mathcal{H}$. Therefore, $\mathcal{A}'$ succeeds in forging a signature in $\mathcal{H}$ with probability $\frac{\epsilon(\lambda)}{n \cdot p(\lambda)}$. Second, we claim that $\mathcal{H}$ is computationally indistinguishable from the actual interaction between $\mathcal{A}$ and $\mathcal{A}'$ (and therefore, $\mathcal{A}$ successfully forges a signature on $m$ with probability at least $\frac{\epsilon(\lambda)}{n \cdot p(\lambda)} - \mathsf{negl}(\lambda)$). Indeed, indistinguishability follows from the psuedorandomness of $F$. (A distinguisher $\mathsf{D}$ can be used to build a distinguisher $\mathsf{D}'$ between a single image of either $F$ or a random function, given $\mathsf{poly}(\lambda)$ images of $F$: it simply uses the images to emulate $\mathcal{A}'$ in the interaction with $\mathcal{A}$, using the challenge image to generate the verification key $\mathsf{vk}$.)

**Standard and special modes are indistinguishable.** By a standard hybrid argument, the indistinguishability of the standard and special modes of the underlying one-time dual-mode signature scheme (we denote these distributions by $\mathcal{D}_{\text{standard},1}, \mathcal{D}_{\text{special},1}$, respectively) implies that $\mathcal{D}_{\text{standard},1}^k \approx \mathcal{D}_{\text{special},1}^k$ for any $k = \mathsf{poly}(\lambda)$, where $\mathcal{D}^k$ denotes $k$ independent samples from $\mathcal{D}$.

We prove indistinguishability by a sequence of hybrids in which the signatures are generated using a signing key in which we iteratively replace the layers from "special" to "standard". That is, for $0 \leq i \leq n+1$, we define $\mathcal{H}_i$ as follows. (1) For levels $n$ up to $i$, generate the verification keys, and

signatures, on the paths leading from messages in $M$ to the root as in the $\mathsf{SpecialGen}$ algorithm (i.e., these are generated iteratively using the $\mathsf{SpecialGen}^1$ algorithm). (2) For levels $i-1$ up to 0, generate the signing and verification keys on these paths using $\mathsf{KeyGen}^1$, and honestly sign the verification keys on the paths. $\mathcal{H}_i$ includes the verification key associated with the root, and the signatures on all messages in $M$. Notice that $\mathcal{H}_0 \equiv \mathcal{D}_{\text{special}}$ and $\mathcal{H}_{n+1} \equiv \mathcal{D}_{\text{standard}}$, so it suffices to prove that for every $1 \leq i \leq n+1$, $\mathcal{H}_i \approx \mathcal{H}_{i-1}$.

Fix some $i$. Notice that the verification keys in levels $i, \cdots, n$, the signing and verification keys in levels $0, \cdots, i-2$, and the signatures in all levels except $i-1, i-2$, are identically distributed in $\mathcal{H}_i, \mathcal{H}_{i-1}$. Therefore, we can fix these values, while preserving the computational distance between the hybrids. Under this fixing, $\mathcal{H}_i, \mathcal{H}_{i-1}$ differ only in the verification key of level $i-1$, and the signatures in levels $i-1, i-2$: in $\mathcal{H}_{i-1}$ the level-$(i-1)$ verification keys and signatures are generated by running $\mathsf{SpecialGen}^1$, whereas in $\mathcal{H}_i$ they are honestly generated using $\mathsf{KeyGen}^1$ and $\mathsf{Sign}^1$. In both hybrids, the level-$(i-2)$ signatures are honestly generated using $\mathsf{Sign}$, but the signed messages contain the level-$(i-1)$ verification keys (on which the hybrids differ). In particular, for some $k = \mathsf{poly}(\lambda)$, $\mathcal{H}_{i-1}$ is efficiently computable from $\mathcal{D}^k_{\text{special},1}$ given the fixed values (one need only generate the level-$(i-2)$ signatures, for which the signing key has been fixed), whereas $\mathcal{H}_i$ is efficiently computable from $\mathcal{D}^k_{\text{standard},1}$ (using the same function and fixed values), and so $\mathcal{H}_i \approx \mathcal{H}_{i-1}$ by the indistinguishability of standard and special modes of the underlying one-time scheme. (That is, given a distinguisher $\mathsf{D}$ between $\mathcal{H}_i, \mathcal{H}_{i-1}$, we construct a distinguisher $\mathsf{D}_1$ between $\mathcal{D}^k_{\text{standard},1}, \mathcal{D}^k_{\text{special},1}$, by hard-wiring all the fixed values, as well as the level-$(i-2)$ signing keys, into $\mathsf{D}_1$. Given the verification keys and signatures for level $i-1$, $\mathsf{D}_1$ uses the signing keys of level $i-2$ to generate the level-$(i-2)$ signatures which, together with the fixed values, constitutes the inputs to $\mathsf{D}$.)

**Special mode cannot sign additional messages.** The argument is similar to the proof that the scheme is existentially unforgeable. Given an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in forging a signature in the special mode with probability $\epsilon(\lambda)$, we construct an adversary $\mathcal{A}'$ that succeeds (with probability $\frac{\epsilon(\lambda)}{n \cdot p(\lambda)} - \mathsf{negl}(\lambda)$) in forging a signature in the underlying one-time signature scheme (where $p(\lambda)$ bounds the running time of $\mathcal{A}$).

$\mathcal{A}'$ operates as follows. First, it obtains from its challenger a verification key $\mathsf{vk}$. Then, it runs $\mathcal{A}_1$ to obtain a list $M$ of messages for which $\mathcal{A}$ requests special-mode signatures, and a state $\mathsf{St}$. Then, $\mathcal{A}'$ picks $i^* \in_R [p(\lambda)]$ and $j^* \in_R \{0, \cdots, n-1\}$. (Intuitively, $\mathcal{A}'$ guesses that the following holds for the message $m'$ on which $\mathcal{A}$ will try to forge a signature: its path in the tree, and the path of the $i^*$'th message in $M$, have the longest intersection (i.e., agree on the longest prefix of a path, out of all messages in $M$), and these paths coincide up to level $j^*$.) Next, $\mathcal{A}'$ emulates $\mathsf{SpecialGen}$ to generate the verification key $\mathsf{vk}_\epsilon$ of the root, and the signatures on the messages in $M$, with the following modification: it uses $\mathsf{vk}$ as the verification key of the level-$(j^*+1)$ node $v$ who is the sibling of the level-$(j^*+1)$ node on the path of the $i^*$'th message in $M$ (instead of choosing it as specified by the $\mathsf{SpecialGen}$ algorithm). Notice that if $\mathcal{A}'$ was wrong in its guess for $i^*, j^*$, then $v$ might appear on the path to some message in $M$ (in which case $\mathcal{A}'$ cannot generate a signature, since it does not know the corresponding signing key). If this happens, $\mathcal{A}'$ aborts the forging game. Then, $\mathcal{A}'$ runs $\mathcal{A}_2$ with $\mathsf{St}$, $\mathsf{vk}_\epsilon$, and these signatures, and obtain a message $m'$, and a purported signature $\sigma' = \left((\sigma_0, \cdots, \sigma_n), \left(\mathsf{vk}_i^0, \mathsf{vk}_i^1\right)_{1 \leq i \leq n}\right)$ on $m'$. Finally, $\mathcal{A}'$ sends $\sigma_{j^*+1}$ as the signature on the message $\left(\mathsf{vk}^0_{j^*+2}, \mathsf{vk}^1_{j^*+2}\right)$ (or, if $j^* = n-1$, on the message $m'$).

We claim that $\mathcal{A}'$ successfully forges a signature (on $\left(\mathsf{vk}^0_{j^*+2}, \mathsf{vk}^1_{j^*+2}\right)$ if $j^* < n-1$, and on $m'$ if $j^* = n-1$) with probability $\frac{\epsilon(\lambda)}{n \cdot p(\lambda)}$. Notice first that the game between $\mathcal{A}'$ and $\mathcal{A}$ is computationally indistinguishable from the actual special-mode forging game of $\mathcal{A}$ (of Definition 5.9). Indeed, the only difference is in the verification key $\mathsf{vk}$ used in the level-$(j^*+1)$ node which is the sibling of the level-$(j^*+1)$ node on the path to the leaf of the $i^*$'th message: in the special-mode forging game this is a

special-mode verification key, whereas in the game with $\mathcal{A}'$ it is an actual (standard-mode) verification key (generated by $\mathsf{KeyGen}^1$). However, these keys are indistinguishable by the indistinguishability of special and standard modes of the underlying one-time scheme and Remark 5.10. (More specifically, given a distinguisher $\mathsf{D}$ between the actual special-mode forging game, and the interaction between $\mathcal{A}, \mathcal{A}'$, one can construct a distinguisher $\mathsf{D}'$ between standard and special-mode verification keys: given (a standard or special-mode) verification key $\mathsf{vk}$, $\mathsf{D}'$ emulates the interaction between $\mathcal{A}, \mathcal{A}'$, using $\mathsf{vk}$ as the challenge verification key. It then runs $\mathsf{D}$ on the transcript obtained in this way, and outputs whatever $\mathsf{D}$ outputs.) Therefore, if $\sigma'$ is a valid signature on $m'$ (which happens with probability $\epsilon(\lambda)$), then in particular $\mathsf{Ver}^1(\mathsf{vk}, \sigma_{j^*+1}, m'') = 1$ (where $m'' = \left(\mathsf{vk}^0_{j^*+2}, \mathsf{vk}^1_{j^*+2}\right)$ if $j^* < n-1$, otherwise $m'' = m'$), i.e., if $\mathcal{A}'$ guessed $i^*, j^*$ correctly, then it successfully forges a signature. □

By applying the transformation of Lemma 5.12 to the one-time dual-mode signatures of Lemma 5.11, we obtain the following result:

**Theorem 5.13.** *Assume that OWPs and CRHFs exist. Then there exists a dual-mode signature scheme. Moreover, there exists a polynomial $p(\lambda)$ such that signatures on length-$n$ messages have length $5p(\lambda) \cdot (n+1)$, signing keys have length $2p(\lambda) + \lambda$, and verification keys have length $2p(\lambda)$.*

### 5.2.2 The MKSE Scheme

In this section we use dual-mode signatures to construct an MKSE scheme based on pc-diO. Concretely, we instantiate Construction 2 with a pc-diO obfuscator and the dual-mode signatures of Theorem 5.13. The properties of the resultant scheme are summarized in the following claim (whose proof is similar to that of Claim 5.4).

**Claim 5.14** (MKSE (Short Share Keys) from pc-diO)**.** *Assume that Construction 2 is instantiated with:*

- *a secure pc-diO obfuscator $\mathcal{O}$ for TMs,*
- *a family of collision-resistent hash functions $h$,*
- *a double symmetric encryption scheme $\left(\mathsf{KeyGen}^2, \mathsf{Enc}^2, \mathsf{Dec}^2\right)$, and*
- *a secure dual-mode signature scheme,*

*then Construction 2 is a secure MKSE.*

*Moreover, if on messages of length $n$ the dual-mode signature scheme outputs signing and verification keys of length $\mathsf{poly}(\lambda)$, and signatures of length $n \cdot \mathsf{poly}(\lambda)$, then the following holds for the MKSE scheme for universe $\mathcal{U}$. Data and query keys have length $\mathsf{poly}(\lambda)$, queries have length $\log|\mathcal{U}| \cdot \mathsf{poly}(\lambda)$, and for a set $S$, its corresponding processed set has size $|S| \cdot \mathsf{poly}(\lambda)$. Furthermore, if: (1) evaluating $h$ on length-$n$ inputs takes $H_T(n)$ time, and outputs a hash of length $H_\ell(n)$; and (2) there exist functions $\mathsf{s}, T_{\mathcal{O}} : \mathbb{N} \to \mathbb{N}$ such that for every TM $M$, $|\mathcal{O}(M)| \leq \mathsf{s}(|M|)$, and running $\mathcal{O}(M)$ on inputs of length $n$ takes $T_{\mathcal{O}}(\mathsf{TIME}(M,n))$ time, where $\mathsf{TIME}(M,n)$ is the running time of $M$ on length-$n$ inputs; then: running $\mathsf{Search}$ on a set $S$ takes $T_{\mathcal{O}}(H_T(|S| \cdot \mathsf{poly}(\lambda)) + |S| \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, \log|\mathcal{U}|))$ time, and share keys have size $\mathsf{s}(H_\ell(|S| \cdot \mathsf{poly}(\lambda)) \cdot \mathsf{poly}(\lambda))$.*

*Proof.* The correctness and complexity of the scheme is proven similarly to Claim 5.4. (The only difference is in the length of queries, which contain a signature on a keyword $w \in \mathcal{U}$, and the running time of $\mathsf{Search}$, which needs to verify the signature. In both cases, the increase in complexity is caused because signatures on $w \in \mathcal{U}$ have length $\log|\mathcal{U}| \cdot \mathsf{poly}(\lambda)$.)

The security proof proceeds in a sequence of hybrids similar to the proof of Claim 5.4, but introduces additional complications due to using a weaker obfuscator primitive. More specifically, let $\mathcal{A}$ be a PPT adversary in the security game of Definition 3.1, and let $\mathsf{view}_b, b \in \{0,1\}$ denote its view in the security game with bit $b$. We define hybrids $\mathcal{H}_0, \mathcal{H}_1$, and $\mathcal{H}_2$ as in the proof of Claim 5.4, and $\mathsf{view}_0 \approx \mathcal{H}_0$ by the same arguments. Indeed, as discussed there, the obfuscated programs in $\mathsf{view}_0, \mathcal{H}_0$ are differing inputs in relation to *every* auxiliary input, and in particular when this auxiliary input is the randomness used by the sampler to sample the programs. $\mathcal{H}_0 \approx \mathcal{H}_2$ because the indistinguiushability argument did not use diO security.

Next, we define a new hybrid $\mathcal{H}'_2$ in which the signatures on the queries $\mathcal{W}^0_j$ of every querier $j$, and his verification key $\mathsf{vk}_j$, are generated using the $\mathsf{SpecialGen}$ algorithm (instead of the $\mathsf{KeyGen}$ and $\mathsf{Sign}$ algorithms). We show that $\mathcal{H}'_2 \approx \mathcal{H}_2$ by the indistinguishability of standard and special modes of the dual-mode signature scheme. We condition both hybrids on the values of the sets $S^0_i, S^1_i$ of data owners, their data keys, the processed sets, the encryption keys of the queriers, the keywords they search for, and their encryptions. (This is possible by an averaging argument, since these values are identically distributed in both hybrids.) Let $m$ denote the number of queriers, then we define a sequence of hybrids $\mathcal{H}^0, \cdots, \mathcal{H}^m$, where in $\mathcal{H}^j$, the signatures and verification key of the first $j$ queriers are generated using $\mathsf{SpecialGen}$, and the signatures and verification keys of all other queriers are honestly generated (using $\mathsf{KeyGen}$ and $\mathsf{Sign}$). We prove that $\mathcal{H}^j \approx \mathcal{H}^{j-1}$ for every $1 \leq j \leq m$, and conclude that $\mathcal{H}_2 = \mathcal{H}^0 \approx \mathcal{H}^m = \mathcal{H}'_2$.

Fix some $j$. Given a distinguisher $\mathsf{D}$ between $\mathcal{H}^j, \mathcal{H}^{j-1}$, we construct a distinguisher $\mathsf{D}'$ (with the same distinguishing advantage) between $\mathcal{D}_{\text{standard}}$ and $\mathcal{D}_{\text{special}}$ (of Definition 5.9), when these are generated for the ciphertexts encrypting the keywords in $\mathcal{W}^0_j$, and conclude that $\mathcal{H}^j \approx \mathcal{H}^{j-1}$ by the indistinguishability of standard and special modes property. We hard-wire into $\mathsf{D}'$ the signing, verification keys, and queries of every $j' \neq j$, as well as all share keys $\Delta_{j',i}$ for $i \in \mathcal{D}$ (this is possible because these values are identically distributed in $\mathcal{H}^j, \mathcal{H}^{j-1}$ and so we can fix them into both hybrids). Given a verification key $\mathsf{vk}$, and a list $L$ of signatures on the ciphertexts of querier $j$, $D'$ generates for every edge $(j,i) \in E$ the program $P_{\mathsf{K},(\mathsf{K}^u,\mathsf{vk}),h(T)}$ (where $\mathsf{K}, \mathsf{K}^u$, and $h(T)$ are taken from the hard-wired values), and uses $\mathcal{O}$ to generate the obfuscated program $\Delta_{j,i}$. Then, $\mathsf{D}'$ generates the queries of querier $j$ by concatenating the corresponding signature to each ciphertext of $j$. Together with the hard-wired values, this gives the entire hybrid, and $\mathsf{D}'$ runs $\mathsf{D}$ on the hybrid, and outputs whatever $\mathsf{D}$ outputs. Notice that if $\mathsf{vk}$ and the signatures were honestly generated, then the input to $\mathsf{D}$ is distributed according to $\mathcal{H}^{j-1}$, otherwise it is distributed according to $\mathcal{H}^j$, so $\mathsf{D}'$ and $\mathsf{D}$ have the same distinguishing advantage.

Next, we define $\mathcal{H}_3$ as in the proof of Claim 5.4 (but notice that the verification keys in every $\Delta_{j,i}$ were generated using $\mathsf{SpecialGen}$), and claim that $\mathcal{H}'_2 \approx \mathcal{H}_3$ by the *pc*-diO security of $\mathcal{O}$. We define the hybrids $\mathcal{H}^0, \cdots, \mathcal{H}^d$ as in the argument that $\mathcal{H}_2 \approx \mathcal{H}_3$ in the proof of Claim 5.4 (except that we use $\mathcal{H}'_2$ instead of $\mathcal{H}_2$), and show that $\mathcal{H}^l \approx \mathcal{H}^{l-1}$ for every $1 \leq l \leq d$.

Fix some $1 \leq l^* \leq d$, and let $(j,i)$ be the edge for which $\Delta^{l^*}$ was generated. We hard-wire the sets $S^0_{i'}, S^1_{i'}, i' \in \mathcal{D}$, and all the keywords that queriers ask about (in both the 0-experiment and the 1-experiment), into $\mathcal{H}^{l^*}, \mathcal{H}^{l^*-1}$ (this is possible because these values are identical in both hybrids). We additionally hard-wire the keys of every querier $j', j' \neq j$ and data owner $i', i' \neq i$, the encrypted sets $T_{i'}, i' \neq i$, the queries of querier $j'$, and the share keys $\Delta_{j',i'}, \Delta'_{j',i'}$ ($\Delta_{j',i'}$ denotes a key in $\mathcal{H}'_2$, $\Delta'_{j',i'}$ denotes a key in $\mathcal{H}_3$). (This is possible because these values are identically distributed in both hybrids.) We show that $\Delta_{j,i}, \Delta'_{j,i}$ sampled in $\mathcal{H}^{l^*-1}, \mathcal{H}^{l^*}$ (respectively) form a public-coin differing-inputs family of TMs (and conclude that $\mathcal{H}^{l^*-1} \approx \mathcal{H}^{l^*}$ by the pc-diO security of $\mathcal{O}$).

Let $\mathcal{P}, \mathcal{P}'$ denote the distributions over programs obfuscated in $\Delta_{j,i}, \Delta'_{j,i}$, and let $\mathsf{D}$ be a PPT algorithm that obtains $(P, P') \leftarrow \mathcal{P} \times \mathcal{P}'$ and $r$, where $r$ is the randomness used to sample $P, P'$. We assume the "worst-case" scenario in which all the values we have fixed into $\mathcal{H}^{l^*}, \mathcal{H}^{l^*-1}$ are known to

$\mathsf{D}$. Notice that from the randomness $r$ of the sampler, $\mathsf{D}$ learns the encryption keys $\mathsf{K}_i, \mathsf{K}_j^u$ (the left component of these keys is needed to generate $P$, whereas the right component is needed to generate $P'$), as well as the encrypted set $T_i$, the verification key $\mathsf{vk}_j^u$ for signatures of querier $j$, and the queries of $j$ (which consist of encryptions of keywords, and signatures on these encryptions; the signatures were generated together with the verification key by $\mathsf{SpecialGen}$). (We note that from these values $\mathsf{D}$ can compute on its own the share keys $\Delta_{j',i}, \Delta'_{j',i}$ for $(j',i) \in E$, and $\Delta_{j,i'}, \Delta'_{j,i'}$ for $(j,i') \in E$.) We show that $\mathsf{D}$ succeeds in finding a differing input only with negligible probability.

Consider first the inputs (for $P, P'$) which $\mathsf{D}$ knows (from the hard-wired values, or what it can deduce from $r$), i.e., the encrypted set $T_i$ (which encrypts the elements of $S_i^0$ in the left components, and the elements of $S_i^1$ in the right components; for $i \in \mathcal{D}_c$ this holds since $S_i^0 = S_i^1$), and the queries of querier $j$. For every such query $q$ there exists a pair $(w_L, w_R)$ of keywords such that $q$ is of the form $q = (c = (c_L, c_R), \sigma)$ where $c_\star \leftarrow \mathsf{Enc}\left(\mathsf{K}^u_{\star,j}, m_\star\right), \star \in \{L, R\}$, $\sigma$ is a valid signature on $c$ (generated by $\mathsf{SpecialGen}$), and $w_L \in S_i^0 \Leftrightarrow w_R \in S_i^1$. In particular, $P(q, T_i) = P'(q, T_i)$ except with negligible probability since except with negligible probability, the checks in Steps (1)-(2) succeed in both cases (by indistinguishability of the standard and special modes of the signature scheme, $\sigma$ is indistinguishable from a valid signature on $c$, which by the correctness of the signature scheme, would pass verification), and Steps (4)-(5) return the same outcome ($P$ searches for $w_L$ in $S_i^0$, since it decrypts using $\mathsf{Dec}^2$, whereas $P'$ searches for $w_R$ in $S_i^1$, since it decrypts with $\mathsf{Dec}^R$ and the right component of $T_i$ encrypts $S_i^1$; and $w_L \in S_i^0 \Leftrightarrow w_R \in S_i^1$).

Next, we claim that for every other possible input $(T', c', \sigma')$ that $\mathsf{D}$ chooses, $P(T', c', \sigma') = P'(T', c', \sigma') = 0$ except with negligible probability. Indeed, if $(c', \sigma') \neq q$ (where $q$ is some query of querier $j$) then by the property that special-mode keys cannot sign additional messages, with overwhelming probability $\sigma'$ is not a valid signature for $c'$, so the check in Step (1) fails in both $P, P'$. Moreover, if $T' \neq T_i$ then by the collision resistance of $h$, with overwhelming probability $h(T') \neq h(T_i)$, so the check in Step (2) fails. Therefore, $P, P'$ are public-coin differing-inputs.

We now define the last set of hybrids. (These hybrids differ from the corresponding hybrids in the proof of Claim 5.4 only in that the verification keys and signatures are generated in the special mode.)

$\mathcal{H}_4$ : In hybrid $\mathcal{H}_4$, the queries $\left(w_{j,1}^1, \cdots, w_{j,k_j}^1\right)$ of every querier $j \in \mathcal{Q}$ are generated using $\mathsf{Enc}^2$ with message $\left(w_{j,l}^1, w_{j,l}^1\right), 1 \leq l \leq k_j$, instead of $\mathsf{Enc}^L$ with message $\left(w_{j,l}^0, w_{j,l}^1\right), 1 \leq l \leq k_j$.

$\mathcal{H}_3 \approx \mathcal{H}_4$ by a similar argument to the one used to prove $\mathcal{H}_1 \approx \mathcal{H}_2$.

$\mathcal{H}_5$ : In hybrid $\mathcal{H}_5$, the encrypted set $T_i$ of every honest data owner $i \notin \mathcal{D}_c$ is generated as the encryption of $\left(S_i^1, S_i^1\right)$ with $\mathsf{Enc}^2$ instead of with $\mathsf{Enc}^L$.

$\mathcal{H}_4 \approx \mathcal{H}_5$ by a similar argument to the one used to prove $\mathcal{H}_0 \approx \mathcal{H}_1$.

$\mathcal{H}_6$ : In hybrid $\mathcal{H}_6$, the obfuscated program for every edge $(j,i) \in E$ is generated as follows: (1) the hard-wired keys are $\mathsf{K} = (\mathsf{K}_{L,i}, \mathsf{K}_{R,i}), \mathsf{K} = \left(\mathsf{K}^u_{L,j}, \mathsf{K}^u_{R,j}\right)$, instead of $\mathsf{K}' = \left(\vec{0}, \mathsf{K}_{R,i}\right), \mathsf{K}^{u\prime} = \left(\vec{0}, \mathsf{K}^u_{R,j}\right)$; and (2) $\mathsf{Dec}^2$ is used for decryption (instead of $\mathsf{Dec}^R$).

We show that $\mathcal{H}_6 \approx \mathcal{H}_5$ follows from the pc-diO security of $\mathcal{O}$ by a standard hybrid argument in which the obfuscated programs are replaced one at a time. When replacing the program for edge $(j,i)$ from $P'$ (in $\mathcal{H}_5$) to $P$ (in $\mathcal{H}_6$), we hard-wire into the PPT $\mathsf{D}$ (which should find a differing input) the sets $S_{i'}^1$ for every $i' \in \mathcal{D}$, the keywords searched for (in the 1-experiment) by all queriers, the encryption keys of all data owners $i', i' \neq i$ and queriers $j', j' \neq j$, the signing, verification keys, and queries of all queriers $j', j' \neq j$, and all encrypted sets $T_{i'}, i' \neq i$. Also, from the randomness $r$ of

the sampler (of $P, P'$), $\mathsf{D}$ learns the encryption keys $\mathsf{K}_i, \mathsf{K}_j^u$, the (special-mode) verification key $\mathsf{vk}_{s,j}^u$, the encryptions of the keywords which querier $j$ searches for, together with the signatures on these ciphertexts, and the encrypted set $T_i$.

We claim that $\mathsf{D}$ finds a differing input only with negligible probability. The argument is similar to that used to prove $\mathcal{H}_3 \approx \mathcal{H}_2'$. The inputs (for $P, P'$) available to $\mathsf{D}$ (from the hard-wired values, and the randomness of the sampler), i.e., the encrypted set $T_i$, and queries $q = (c, \sigma)$ of querier $j$, where $c \leftarrow \mathsf{Enc}^2\left(\mathsf{K}_j^u, m\right)$ for some $m$, and $\sigma$ is a signature for $c$ (generated using $\mathsf{SpecialGen}$), are not differing-inputs (even though $P'$ decrypts the right component of $c$, whereas $P$ decrypts the left component) because both components of $c$ encrypt $m$ according to $\mathsf{Enc}$ (so both $P, P'$ search for $m$ in $S_i^1$), where $\mathsf{Enc}$ is the encryption scheme underlying $\mathsf{Enc}^2, \mathsf{Enc}^L, \mathsf{Enc}^R$. For every other possible input $(T', c', \sigma')$ that $\mathsf{D}$ chooses, $P\left(T', c', \sigma'\right) = P'\left(T', c', \sigma'\right) = 0$ except with negligible probability: if $(c', \sigma') \neq q$ (where $q$ is some query of querier $j$) then with overwhelming probability $\sigma'$ is not a valid signature on $c'$ (by the property that special-mode signatures cannot sign additional messages); whereas if $T' \neq T_i$ then with overwhelming probability $h\left(T'\right) \neq h\left(T_i\right)$ (by the collision resistance of $h$).

In our final new hybrid $\mathcal{H}_6'$, the signatures on the queries $\mathcal{W}_j^0$ of every querier $j$, and his verification key $\mathsf{vk}_j$, are honestly generated (using the $\mathsf{KeyGen}$ and $\mathsf{Sign}$ algorithms). Then $\mathsf{view}_1 = \mathcal{H}_6' \approx \mathcal{H}_6$ by the same arguments used to show $\mathcal{H}_2' \approx \mathcal{H}_2$. □

Theorem 5.8 now follows as a corollary from Claim 5.14 and Theorem 5.13, and since the existence of a symmetric encryption scheme follows from the existence of a CRHF.

# 6 Extensions and Open Problems

In this section we discuss possible extensions of our MKSE definition and constructions, and point out a few open problems in the field.

We have focused on a *selective, indistinguishability-based* MKSE notion (Definition 3.1). One could also consider several other formulations, as we now discuss.

**Simulation-based security.** First, one can consider a *selective simulation-based notion*, in which the real-world view of any PPT adversary can be efficiently simulated given only "minimal" information. More specifically, at the onset of the execution the adversary chooses (as in Definition 3.1) sets of queriers, data owners, and corrupted data owners; a share graph; keyword sets for all data owners; data keys for corrupted data owners; and a (possibly empty) set of distinct keyword queries for each querier. The simulator is then given the sets and data keys of corrupted data owners; the *sizes* of the sets of honest data owners; the share graph; and for each keyword query $w$ of querier $j$, and every edge $(j, i)$ in the graph, whether $w \in S_i$ or not (where $S_i$ is the set of data owner $i$). The simulator then generates a complete simulated adversarial view, namely processed sets for all data owners, share keys for all edges in the share graph, and queries for every keyword query. Intuitively, we say that an MKSE is *simulation-secure* if for every PPT adversary there exists a PPT simulator as above, such that the real and simulated views are computationally indistinguishable. The PRF-based MKSE (Construction 1) is simulation-secure (see Remark 4.4), and we leave it as an open problem to determine whether the MKSE with short share keys (Construction 2, based on diO or pc-diO) is simulation-secure.

A natural question that arises in this context is whether indistinguishability-based security (as in Definition 3.1) implies simulation-based security (as outlined above). One approach towards tackling this question is to describe an algorithm that, given the input of the simulator (as specified above), generates an assignment for the sets of the honest data owners, and for all keyword queries, in a way that is consistent with the outcome of searching for these keywords. Concretely, this approach

reduces the task of constructing a simulator to the following graph problem: given a bipartite graph $G = (L, R, E)$; a set $\{n_v \ : \ v \in R\}$ of natural numbers; and for every $u \in L$, a set of coloring of the edges touching $u$ in two colors (blue and red), assign a set $S_v \subseteq \mathcal{U}$ to every $v \in R$ (recall that $\mathcal{U}$ is a universe of possible keywords), and a value $w_u^c \in \mathcal{U}$ to every $u \in L$ and every coloring $c$ of the edges that touch $u$, such that the following holds:

1. for every $v \in R$, $|S_v| = n_v$,
2. for every $u \in L$, and two colorings $c_1, c_2$ of the edges touching $u$, $w_u^{c_1} \neq w_u^{c_2}$, and
3. for every $u \in L$, every coloring $c$ of the edges that touch $u$, and every edge $(u, v) \in E$, $c\,(u, v) =$ blue if and only if $w_u^c \in S_v$, where $c\,(u, v)$ is the color of the edge $(u, v)$ in the coloring $c$.

(We note that Item 1 guarantees that sets have the "right" size; Item 2 guarantees that the queries made by each querier are distinct; and Item 3 guarantees that the assignments to the sets and keywords searched for are consistent.) At a high level, the main challenge is in finding an assignment for the set that would be consistent over the queries of *multiple* queriers, while simultaneously satisfying the restriction on the size of the set. Intuitively, this issue does not arise in the PRF-based construction since each share key $\Delta_{j,i}$ encodes the entire set, and the $\mathsf{Search}$ algorithm does not use the processed set at all (so issues of consistency across *different* queriers do not arise).

**Adaptive security.** Another possible dimension of generalizing Definition 3.1 is to consider an (either indistinguishability-based or simulation-based) adaptive definition. At a high level, in this setting the adversary may adaptively generate queriers, data owners (with their sets and data keys), edges in the share graph, and keyword queries, and immediately receives the resultant values. (For example, when an adversary specifies a new data owner and his data key and set, he receives the corresponding processed set; when he adds an edge to the share graph, he receives the corresponding share key, etc.) The security requirement should hold as long as at the end of the execution, the data sets, share graph, and queries satisfy the restrictions imposed by the (selective) security definition (in a simulation-based definition, the only restriction is that the queries of each querier are distinct; in an indistinguishability-based definition there are further restrictions as specified in Definition 3.1).

Natural approaches towards proving adaptive security, even for the PRF-based construction (Construction 1), seem to run into "selective opening type" issues: in the security proof, we would naturally want to replace the pseudorandom images of the PRF $F$ in share keys with random values, however we do not a-priori know which values the adversary will ask to be "opened" (by making a keyword query for a keyword in the set corresponding to the share key; recall that these queries constitute a key for $F$). Consequently, we cannot a-priori determine which values should remain pseudorandom (so that they can later be opened). However, we can show that Construction 1 is adaptively simulation-secure *in the Random Oracle model*, namely when all evaluations of $F$ are replaced with calls to the random oracle (replacing $F_K\,(x)$ with a call to $\mathsf{RO}\,(K, x)$). The random oracle circumvents such "selective opening type" issues since any (randomly assigned) output of the random oracle can later be "explained" by consistently assigning the random oracle outputs at other (related) points.

**Efficiency and security tradeoffs.** Finally, an interesting avenue for future research is exploring the tradeoffs between efficiency of the MKSE scheme, and the underlying assumptions. Our MKSE with short share keys (Construction 2) indicates the hardness of proving an unconditional lower bound on the size of share keys, since it would require ruling out the existence of diO for a *specific* class of samplers. However, it does not rule out the possibility of constructing an MKSE scheme with short share keys based on weaker assumptions (such as the existence of iO for TMs, or ideally, on the existence of OWFs). More generally, one could ask how the search time, and size of share keys, relate to each other; and if there is a lower bound on "search time plus share key size".

## Acknowledgments

We thank the anonymous PKC reviewers for suggesting to use hashing to reduce search time to $O(1)$ in Construction 1. This work was supported by NSF grants CNS-1314722, CNS-1413964, TWC-1664445 and TWC-1646671. The third author was supported in part by The Eric and Wendy Schmidt Postdoctoral Grant for Women in Mathematical and Computing Sciences.

## References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/2013/689.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BHJP14] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, August 2014.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *EUROCRYPT 2016, Proceedings, Part II*, pages 792–821, 2016.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 79–88. ACM Press, October / November 2006.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 668–679. ACM Press, October 2015.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*. The Internet Society, February 2014.

[CLH+14] Xiaofeng Chen, Jin Li, Xinyi Huang, Jingwei Li, Yang Xiang, and Duncan S. Wong. Secure outsourced attribute-based signatures. *IEEE Trans. Parallel Distrib. Syst.*, 25(12):3285–3294, 2014.

[CLW16] Baojiang Cui, Zheli Liu, and Lingyu Wang. Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage. *IEEE Transactions on computers*, 65(8):2374–2385, 2016.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05*, volume 3531 of *LNCS*, pages 442–455. Springer, Heidelberg, June 2005.

[DRD11] Changyu Dong, Giovanni Russello, and Naranker Dulay. Shared and searchable encrypted data for untrusted servers. *Journal of Computer Security*, 19(3):367–397, 2011.

[FKS82] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. In *FOCS 1982*, pages 165–169, 1982.

[FVY+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. SoK: Cryptographically protected database search. *arXiv preprint arXiv:1703.02014*, 2017.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *CRYPTO 2014, Proceedings, Part I*, pages 518–535, 2014.

[GMN+16] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1353–1364. ACM Press, October 2016.

[Goh03] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216.

[Gol01] Oded Goldreich. The Foundations of Cryptography - Volume 1, Basic Techniques. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press, 2004.

[GSB+17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy*, pages 655–672. IEEE Computer Society Press, May 2017.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 668–697. Springer, Heidelberg, March 2015.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1329–1340. ACM Press, October 2016.

[KOR+16] Aggelos Kiayias, Ozgur Oksuz, Alexander Russell, Qiang Tang, and Bing Wang. Efficient encrypted keyword search for multi-user data sharing. In Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows, editors, *ESORICS 2016, Part I*, volume 9878 of *LNCS*, pages 173–195. Springer, Heidelberg, September 2016.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 965–976. ACM Press, October 2012.

[Lam] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International Palo Alto.

[LCL+14] Jin Li, Xiaofeng Chen, Mingqiang Li, Jingwei Li, Patrick PC Lee, and Wenjing Lou. Secure deduplication with efficient and reliable convergent key management. *IEEE transactions on parallel and distributed systems*, 25(6):1615–1625, 2014.

[LLC+12] Jingwei Li, Jin Li, Xiaofeng Chen, Chunfu Jia, and Zheli Liu. Efficient keyword search over encrypted data with fine-grained access control in hybrid cloud. In *Network and System Security - 6th International Conference, NSS 2012*, pages 490–502, 2012.

[LWC+13] Zheli Liu, Zhi Wang, Xiaochun Cheng, Chunfu Jia, and Ke Yuan. Multi-user searchable encryption with coarser-grained access control in hybrid cloud. In *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, 2013*, pages 249–255, 2013.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 644–655. ACM Press, October 2015.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC 1989*, pages 33–43, 1989.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC 1990*, pages 427–437, 1990.

[PSV+14] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 157–172, 2014.

[PW16] David Pouliot and Charles V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1341–1352. ACM Press, October 2016.

[PZ13] Raluca Ada Popa and Nickolai Zeldovich. Multi-key searchable encryption. Cryptology ePrint Archive, Report 2013/508, 2013. http://eprint.iacr.org/2013/508.

[RMÖ17] Cédric Van Rompay, Refik Molva, and Melek Önen. A leakage-abuse attack against multi-user searchable encryption. *PoPETs*, 2017(3):168, 2017.

[SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*. The Internet Society, February 2014.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.

[Tan14] Qiang Tang. Nothing is for free: security in searching shared and encrypted data. *IEEE Transactions on Information Forensics and Security*, 9(11):1943–1952, 2014.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 707–720, 2016.

[ZNS11] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. Multi-user keyword search scheme for secure data sharing with fine-grained access control. In *International Conference on Information Security and Cryptology*, pages 406–418. Springer, 2011.

# A Obfuscation Definitions

In this section we formally define the notions of program obfuscation which we use: differing-inputs obfuscation (diO) [BGI+01], and public-coin differing-inputs obfuscation (pc-diO) [IPS15]. Throughout the section, we use $\mathsf{steps}(M, x)$ to denote the number of steps the Turing Machine $M$ takes when running on input $x$. We first define differing-inputs Turing Machine (TM) families.

**Definition A.1** (Differing-inputs TM family [BGI+01, ABG+13])**.** A family $\mathcal{M}$ of Turing Machines, associated with an efficient sampler $\mathsf{Sampler}_{\mathcal{M}}$, is a *Differing-Inputs Turing Machine Family* if the output of $\mathsf{Sampler}_M$ is a pair of Turing machines $(M_0, M_1) \in \mathcal{M} \times \mathcal{M}$ such that $|M_0| = |M_1|$, and for every (possibly non-uniform) polynomial adversary $\mathcal{A}$, there exists a negligible function $\epsilon$ such that for any $\lambda$:

$$\Pr\left[\begin{array}{c} M_0(x) \neq M_1(x) \wedge \\ \mathsf{steps}(M_0, x) = \mathsf{steps}(M_1, x) = t \end{array} \;:\; \begin{array}{c} (M_0, M_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}_M\left(1^\lambda\right) \\ (x, 1^t) \leftarrow \mathcal{A}\left(1^\lambda, M_0, M_1, \mathsf{aux}\right) \end{array}\right] \leq \epsilon(\lambda)$$

**Definition A.2** (Differing-inputs Obfuscator for TMs [BGI+01, ABG+13])**.** A uniform PPT machine $\mathcal{O}$ is a *differing-inputs Obfuscator (diO)* for a differing-inputs Turing Machine family $\mathcal{M}$ if the following holds:

- **Correctness:** for all security parameters $\lambda \in \mathcal{N}$, $M \in \mathcal{M}$, and inputs $x$,

$$\Pr\left[M'(x) = M(x) \;:\; M' \leftarrow \mathcal{O}(1^\lambda, M)\right] = 1.$$

- **Security:** for any (possibly non-uniform) PPT distinguisher $\mathsf{D}$ there exists a negligible function $\epsilon$ such that for any $\lambda$

$$\begin{aligned} &|\Pr\left[\mathsf{D}\left(M', \mathsf{aux}\right) = 1 \;:\; (M_0, M_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}_{\mathcal{M}}\left(1^\lambda\right), M' \leftarrow \mathcal{O}\left(1^\lambda, M_0\right)\right] \\ &- \Pr\left[\mathsf{D}\left(M', \mathsf{aux}\right) = 1 \;:\; (M_0, M_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}_{\mathcal{M}}\left(1^\lambda\right), M' \leftarrow \mathcal{O}\left(1^\lambda, M_1\right)\right]| \end{aligned}$$

  is at most $\epsilon(\lambda)$.

- **Polynomial blowup:** there exists a (global) polynomial $p : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all security parameters $\lambda \in \mathcal{N}$, and all $M \in \mathcal{M}$, $|M'| \leq p(\lambda, |M|)$, where $M' \leftarrow \mathcal{O}\left(1^\lambda, M\right)$.

Next, we define the notion of *public-coin* diO. We first define public-coin differing-inputs samplers for Turing Machines.

**Definition A.3** (Public-coin differing-inputs sampler for TMs [IPS15])**.** An efficient non-uniform sampler $\mathsf{Sampler}_{\mathcal{M}}$ is a *Public-Coin Differing-Inputs sampler* for a family $\mathcal{M}$ of Turing Machines if the output of $\mathsf{Sampler}_M$ is a pair of Turing machines $(M_0, M_1) \in \mathcal{M} \times \mathcal{M}$ such that $|M_0| = |M_1|$, and for every (posssibly non-uniform) polynomial adversary $\mathcal{A}$, there exists a negligible function $\epsilon$ such that for any $\lambda$:

$$\Pr_r\left[\begin{array}{c} M_0(x) \neq M_1(x) \wedge \\ \mathsf{steps}(M_0, x) = \mathsf{steps}(M_1, x) = t \end{array} \;:\; \begin{array}{c} (M_0, M_1) \leftarrow \mathsf{Sampler}_M\left(1^\lambda, r\right) \\ (x, 1^t) \leftarrow \mathcal{A}\left(1^\lambda, r\right) \end{array}\right] \leq \epsilon(\lambda)$$

**Definition A.4** (Public-Coin Differing-inputs Obfuscator for TMs [IPS15])**.** A uniform PPT machine $\mathcal{O}$ is a *public-coin Turing machine differing-inputs obfuscator* for the family $\mathcal{M}$ of Turing Machines, if the following conditions are satisfied:

- **Correctness:** for all security parameters $\lambda \in \mathcal{N}$, $M \in \mathcal{M}$, and inputs $x$,

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

- **Security:** for any (possibly non-uniform) PPT distinguisher $\mathsf{D}$ there exists a negligible function $\epsilon$ such that for any $\lambda$

$$\begin{aligned} &|\Pr\left[\mathsf{D}\left(M', r\right) = 1 \ : \ (M_0, M_1) \leftarrow \mathsf{Sampler}_{\mathcal{M}}\left(1^\lambda, r\right), M' \leftarrow \mathcal{O}\left(1^\lambda, M_0\right)\right] \\ &- \Pr\left[\mathsf{D}\left(M', r\right) = 1 \ : \ (M_0, M_1) \leftarrow \mathsf{Sampler}_{\mathcal{M}}\left(1^\lambda, r\right), M' \leftarrow \mathcal{O}\left(1^\lambda, M_1\right)\right]| \end{aligned}$$

  is at most $\epsilon(\lambda)$, where the probability is taken over $r$ and the coins of $\mathcal{O}$.

- **Polynomial blowup:** there exists a (global) polynomial $p : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all security parameters $\lambda \in \mathcal{N}$, and all $M \in \mathcal{M}$, $|M'| \leq p\left(\lambda, |M|\right)$, where $M' \leftarrow \mathcal{O}\left(1^\lambda, M\right)$.