New Approaches to Traitor Tracing with Embedded Identities

Rishab Goyal
UT Austin[*]

Venkata Koppula
Weizmann Institute of Science[†]

Brent Waters
UT Austin and
NTT Research[‡]

Abstract

In a traitor tracing (TT) system for n users, every user has his/her own secret key. Content providers can encrypt messages using a public key, and each user can decrypt the ciphertext using his/her secret key. Suppose some of the n users collude to construct a pirate decoding box. Then the tracing scheme has a special algorithm, called Trace, which can identify at least one of the secret keys used to construct the pirate decoding box.

Traditionally, the trace algorithm output only the 'index' associated with the traitors. As a result, to use such systems, either a central master authority must map the indices to actual identities, or there should be a public mapping of indices to identities. Both these options are problematic, especially if we need public tracing with anonymity of users. Nishimaki, Wichs, and Zhandry (NWZ) [Eurocrypt 2016] addressed this problem by constructing a traitor tracing scheme where the identities of users are embedded in the secret keys, and the trace algorithm, given a decoding box D, can recover the entire identities of the traitors. We call such schemes 'Embedded Identity Traitor Tracing' schemes. NWZ constructed such schemes based on adaptively secure functional encryption (FE). Currently, the only known constructions of FE schemes are based on nonstandard assumptions such as multilinear maps and iO.

In this work, we study the problem of embedded identities TT based on standard assumptions. We provide a range of constructions based on different assumptions such as public key encryption (PKE), bilinear maps and the Learning with Errors (LWE) assumption. The different constructions have different efficiency trade offs. In our PKE based construction, the ciphertext size grows linearly with the number of users; the bilinear maps based construction has sub-linear ($\sqrt{n}$) sized ciphertexts. Both these schemes have public tracing. The LWE based scheme is a private tracing scheme with optimal ciphertexts (i.e., $\log(n)$). Finally, we also present other notions of traitor tracing, and discuss how they can be build in a generic manner from our base embedded identity TT scheme.

1 Introduction

Traitor tracing (TT) systems, as introduced by Chor, Fiat, and Naor [CFN94], studied the problem of identifying the users that contributed to building a rogue decoder in a broadcast environment. In a TT system an authority runs a setup algorithm on input a security parameter λ, and the number of users n in the system. This results in generation of a global public key pk, a tracing key key, and n private user keys $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$. Each private key is distributed to an authorized user in the system with the guarantee that it can be used to decrypt any ciphertext ct encrypting a message m under the global public key pk. The first security property satisfied by such systems is that the message will be hidden from every unauthorized user, that is one who does not have access to any secret key. The most salient feature of a traitor tracing system is the presence of an additional tracing algorithm which is used to identify corrupt/coerced users. Suppose an attacker corrupts some subset $S \subseteq \{1, \ldots, n\}$ of authorized users and produces a special decryption

[*]Email: `rgoyal@cs.utexas.edu`. Supported by IBM PhD Fellowship.
[†]Email: `venkata.koppula@weizmann.ac.il`.
[‡]Email: `bwaters@cs.utexas.edu`. Supported by NSF CNS-1908611, CNS-1414082, DARPA SafeWare, Packard Foundation Fellowship, and Simons Investigator Award.

algorithm/device D that can decrypt the ciphertexts with some non-negligible probability. The tracing property of the system states that the tracing algorithm, on input the tracing key key and oracle access to device D, outputs a set of users T where T contains at least one user from the colluding set S (and no users outside of S).

The initial traitor tracing systems allowed bounded collusions [CFN94, SW98, KD98, BF99, CFNP00, SSW01, KY02a, KY02b, PST06, CPP05, ADM+07, FNP07, BP08, LPSS14, ABP+17]; here we focus on collusion resistant systems [BSW06, BW06, GKSW10, Fre10, BZ14, NWZ16, GKW18, CVW+18]. While the concept of traitor tracing was originally motivated by catching corrupt users in broadcast systems, the notion of traitor tracing has numerous other applications such as transmitting sensitive information to first responders (or military personnel etc) on an ad-hoc deployed wireless network, accessing and sharing encrypted files on untrusted cloud storage etc. This propels us to study the problem of traitor traitor more finely with a dedicated focus on understanding the issues that prevent a wider adoptability of such systems.

One major hurdle is that, as per the traditional description of the problem, the tracing portion (that is identifying the corrupt users) is inherently tied to the central authority (key generator) in the system. This is due to the fact that the authority needs to keep track of the users who have been issued private keys, and thus it needs to maintain an explicit mapping (as a look-up table) between the user identification information and the indices of their respective private keys. Otherwise, the output of the tracing algorithm will simply be a subset T of the user indices which can not be linked to actual users in the system, thereby introducing the problem of accountability and circumventing the whole point of tracing traitors. In addition, this not only constrains the authority to be fully stateful (with the state size growing linear with the number of users) by necessitating that the authority *must record* the user information to key index mapping, but also restricts the authority to be the only party which can perform any meaningful notion of tracing if (authorized) user privacy/anonymity is also desired.[1] Therefore, even if the TT system achieves public traceability, that is the tracing key key can be included as part of public parameters, no third party would be able to identify traitors in system due to lack of a public mapping as described above.

Furthermore, in certain situations the user information to key index mapping might be undetermined. For example, suppose all the users in the system obtain their private decryption keys without revealing any sensitive identification information to the key generating authority. (Note that this can be achieved by some sort of two party computation-based transfer between the user and authority.) In such a scenario, it is not clear how tracing would work since the authority would not be able to point to any user in the system as a traitor because the key index to user identity mapping is unknown, even if the tracing algorithm correctly outputs an index of some coerced secret key.

These observations lead to the following question —

Is it possible to embed the user identification information in the private decryption keys such that during tracing the algorithm not only finds the corrupted key indices, but also extracts the corresponding user identities from the pirate decoding device?

Formally, this is captured by giving an additional parameter κ as an input to the setup algorithm, where κ denotes the length of the user identities that can be embedded in the private keys. The setup now outputs a master secret key msk, instead of n private user keys, where msk is used to generate private keys $\mathsf{sk}_{i,\mathsf{id}}$ for any index-identity pair $(i, \mathsf{id}) \in [n] \times \{0,1\}^\kappa$. And the tracing algorithm outputs a set of 'user identities' $T \subseteq \{0,1\}^\kappa$ where $\mathsf{id} \in T$ indicates that id was one of the corrupted users.[2] This interpretation of traitor tracing resolves the above issues of statefulness, third-party traceability, and maintaining a private look-up table for providing user anonymity.

The above-stated question of traitor tracing with embedded information in secret keys was first studied by Nishimaki, Wichs, and Zhandry [NWZ16]. Their approach was to directly work with the existing private linear broadcast encryption (PLBE) framework [BSW06], however that resulted in solutions based

[1] Although the problem of statefulness can be avoided by posting the identity of all authorized users along with their respective (decryption key) indices on a public-bulletin board, such a solution is particularly undesirable in practice as the user identities might include highly sensitive information such as passport information, driving license number, etc.

[2] Note that the tracing algorithm could be additionally asked to output the corresponding user index along with the identity, but since the index $i \in [n]$ could itself be encoded in the identity id using only $\log(n)$ bits therefore this seems unnecessary.

on non-standard assumptions. Concretely, they assume existence of an adaptively-secure collusion-resistant public-key functional encryption (FE) scheme with compact ciphertexts. Currently all known instantiations are either based on multilinear maps [GGH13, CLT13, GGH15, CLT15], or indistinguishability obfuscation [BGI+01, BGI+12]. An important open question here is whether the above problem of embedded information traitor tracing can be solved from standard assumptions such as one-way functions, bilinear assumptions, learning with errors etc. In this work, we study this question and provide a general framework for solving this problem with a wide range of parameter choices and assumption families.

Our Results. We give new constructions for traitor tracing systems with embedded identity tracing under the following assumptions.[3]

Public-key encryption. Our first construction is that of an embedded identity TT scheme with public traceability that relies only on regular PKE schemes. The ciphertext size and length of public key grows linearly in both the number of users n as well as the length of embedded identities κ. This is a natural generalization of the basic TT scheme based on PKE, and is provided to serve as a baseline benchmark for comparing efficiency with other instantiations.

Bilinear maps. Second, we show that using a more algebraic approach via bilinear maps we can build an embedded identity TT scheme with a square-root speed-up w.r.t. the PKE-based scheme. Concretely, the size of ciphertexts and length of public key grows linearly in $\sqrt{n}$ and $\sqrt{\kappa}$. And the scheme still achieves public traceability.

Learning with errors. Lastly, we build a *compact* embedded identity TT scheme secure under the learning with errors (LWE) assumption. Here compactness means that the size of ciphertexts and public key scales polynomially with $\log(n)$ and κ. On the flip side, the tracing key needs to be private, that is it only achieves private key traceability.

These are summarized in Table 1. In the next section we elaborate more on our framework and general methodology for breaking down the problem. Below we discuss our results in more detail.

Assumption	$\lvert\mathsf{ct}\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	Tracing Mode	Unbounded
PKE	$n \cdot \kappa \cdot \mathsf{poly}(\lambda)$	$n \cdot \kappa \cdot \mathsf{poly}(\lambda)$	$\kappa \cdot \mathsf{poly}(\lambda)$	Public	No
Bilinear	$\sqrt{n \cdot \kappa} \cdot \mathsf{poly}(\lambda)$	$\sqrt{n \cdot \kappa} \cdot \mathsf{poly}(\lambda)$	$\log n + \kappa + \mathsf{poly}(\lambda)$	Public	No
LWE	$(\log n + \kappa) \cdot \mathsf{poly}(\lambda)$	$\mathsf{poly}(\lambda)$	$(\log n + \kappa) \cdot \mathsf{poly}(\lambda)$	Private	Yes

Table 1: Embedded Identity Traitor Tracing.

In this work, we provide three new pathways for realizing embedded identity TT systems, and notably the first constructions relying only on standard assumptions. Our first two constructions from public-key encryption and bilinear maps are novel, where our bilinear map based scheme draws ideas from the trace and revoke scheme of Boneh-Waters [BW06]. And, for building an LWE-based solution we adapt the recently introduced Mixed Functional Encryption (Mixed FE) schemes [GKW18, CVW+18] in our framework to get the desired results.

Furthermore, a very important and useful piece of our approach is that it allows us to avoid subexponential security loss in the transformation (due to complexity leveraging) if we allow an *exponential* number of users in the system and the intermediate primitives used are only *selectively-secure*. Particularly, this is used in our LWE-based solution which relies on mixed FE for which most of the current constructions are only known to achieve selective security. (For example, the first mixed FE construction by Goyal, Koppula, and Waters [GKW18] and two of three follow-up constructions by Chen et al. [CVW+18] were proven to be only selectively-secure.) Therefore, our approach also answers the question whether adaptivity is necessary

[3] Nishimaki, Wichs, and Zhandry [NWZ16] used the term “flexible” traitor tracing to refer to schemes where the space of identities that can be traced is exponential. Here we call such TT systems as embedded identity TT schemes (or EITT for short).

for building embedded identity TT schemes if the system is required to support an unbounded number of users. Note that in the prior work of Nishimaki, Wichs, and Zhandry [NWZ16], it was crucial that they start with an 'adaptively-secure' FE scheme for security purposes, but here our approach helps in bypassing the adaptivity requirement. Next, we provide a detailed technical overview of our results.

2 Technical Overview

We start by formally defining the notion of embedded identity traitor tracing (EITT) systems. In order to capture a broader class of traitor tracing systems, we consider three different variants for embedded identity tracing — (1) indexed EITT, (2) bounded EITT, and (3) full (unbounded) EITT. Although the notion of full/unbounded EITT is the most general notion we define and therefore it is also likely the most desirable notion, we believe that both indexed and bounded EITT systems will also find many direct applications as will be evident later during their descriptions. In addition, we also show direct connections between all three notions by providing different transformations among these notions.

Next, we move on to realizing these EITT systems under standard assumptions. To that end, we first introduce a new intermediate primitive which we call *embedded-identity private linear broadcast encryption* (EIPLBE) that we eventually use to build EITT schemes. As the name suggests, the notion of EIPLBE is inspired by and is an extension of *private linear broadcast encryption* (PLBE) schemes introduced in the work of Boneh, Sahai, and Waters (BSW) [BSW06]. BSW introduced the notion of PLBE schemes as a stepping stone towards building general TT systems. In this work, we show that the above-stated extension of PLBE systems can be very useful in that it leads to new solutions for the embedded identity traitor tracing problem.

Finally, we provide multiple instantiations of EIPLBE schemes that are secure under a variety of assumptions (PKE, Bilinear, and LWE). Using these EIPLBE schemes in the aforementioned transformation, we can build various EITT systems with appropriate efficiency metrics.

2.1 Embedded Identity Traitor Tracing Definitions

Let us first formally recall the notion of standard traitor tracing (i.e., without embedding identities in the secret keys). A traitor tracing system consists of four poly-time algorithms — Setup, Enc, Dec, and Trace. The setup algorithm takes as input security parameter λ, and number of users n and generates a public key pk, a tracing key key, and n private keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$. The encryption algorithm encrypts a message m using public key pk, and the decryption algorithm decrypts a ciphertext using any one of the private keys sk_i. The tracing algorithm takes tracing key key, two messages m_0, m_1 as input, and is given (black-box) oracle access to a pirate decoding algorithm D.[4] It outputs a set $S \subseteq [n]$ of users signalling that the keys sk_j for $j \in S$ were used to create the pirate decoder D. The security requirements are as described in the previous section.

Let us now look at how to embed identities in the private user keys such that the tracing algorithm outputs a set of identities instead. Below we describe the identity embedding abstractions considered in this work. Throughout this sequel, κ denotes the length of identities embedded (that is, identity space is $\{0,1\}^\kappa$).

Indexed EITT. We begin with indexed EITT as the simplest way to introduce identity embedding functionality in the standard TT framework is as follows. The setup algorithm takes both n and κ as inputs and outputs a master secret key msk. Such systems will have a special key generation algorithm that takes as input msk along with an index-identity pair $(i, \mathsf{id}) \in [n] \times \{0,1\}^\kappa$, and outputs a user key $\mathsf{sk}_{i,\mathsf{id}}$. When the i^{th} user requests a key then it can supply its identity id, and the authority runs key generation on corresponding inputs to sample a secret key for that particular user.

[4]Traditionally, the tracing algorithm was defined to work only if the decoder box could decrypt encryptions of random messages. However, as discussed in [GKRW18], this definition does not capture many practical scenarios. Therefore we work with a broader abstraction where the trace algorithm works even if the decoder can only distinguish between encryptions of two specific messages.

Encryption, decryption, and tracing algorithms remain unaffected with the exception that the tracing algorithm outputs a set of user identities $S \subseteq \{0,1\}^\kappa$ instead.[5] Now the IND-CPA and secure tracing requirements very naturally extend to indexed EITT systems with one caveat that the adversary can only obtain a user key for each index at most once in the traitor tracing game. Comparing this with standard TT schemes in which each corrupted user receives a unique private key depending on its index, this constraint on set of corruptible keys is a natural translation.

Looking carefully at the above abstraction, we observe that using such indexed systems in practice would seem to resolve the 'look-up table' problem thereby allowing third party tracing, but the problem of statefulness is not yet completely resolved. Concretely, the key generating authority still needs to maintain a counter (that is $\log(n)$ bits) which represents the number of keys issued until that point. Basically each time someone queries for a secret key for identity id, the authority generates a secret key for identity id and index being the current counter value, and it increments the counter in parallel. This constraint stems from the fact that for guaranteeing correct tracing it is essential that the adversary receives at most one key per index $i \in [n]$. Although for a lot of applications indexed EITT might already be sufficient, it is possible that for others this is still restrictive. To that end, we define another EITT notion to completely remove the state as follows.

Bounded EITT. The idea behind bounded EITT is that now the input n given to the setup algorithm represents an upper bound on the number of keys an adversary is allowed to corrupt while the system still guarantees correct traceability. And importantly, the key generation algorithm now only receives an identity id as input instead of an index-identity pair. Thus, the authority does not need to maintain the counter, that is it does not need to keep track of number of users registered. Another point of emphasis is that in a Bounded EITT system if the number of keys an attacker corrupts exceeds the setup threshold n, the attacker may avoid being traced; however, even in this scenario tracing procedure will not falsely indict an non-colluding user. In addition to being a useful property in its own right, the non-false indictment property will be critical in amplifying to Unbounded EITT.

Interestingly, we show a generic transformation from any indexed EITT scheme to a bounded EITT scheme with only a minor efficiency loss. More details on this transformation are provided towards the end of this section. Looking ahead, this transformation only relies on the existence of signatures additionally.

Unbounded EITT. Lastly the most general notion of embedded identity traitor tracing possible is of systems in which the setup algorithm only takes κ the length of identities as input, thus there is no upper bound on the number of admissible corruptions set during setup time. Therefore, the adversary can possibly corrupt an arbitrary (but polynomial) number of users in the system. In this work, we additionally provide an efficient unconditional transformation from bounded EITT schemes to unbounded EITT schemes thereby completely solving the embedded identity tracing problem. More details on this transformation are also provided towards the end of this section.

Next, we move on to building the indexed EITT schemes under standard assumptions. As discussed before, we first introduce the intermediate notion of EIPLBE.

2.2 Embedded-Identity Private Linear Broadcast Encryption

Let us start by recalling the notion of private linear broadcast encryption (PLBE) [BSW06]. Syntactically, a PLBE scheme is same as a traitor tracing scheme as in it consists of setup, key generation, encryption, decryption algorithms with the exception that instead of tracing algorithm it provides an additional encryption algorithm usually referred to as *index-encryption* algorithm. In PLBE systems, the setup algorithm outputs a public, master secret, and index-encryption key tuple $(\mathsf{pk}, \mathsf{msk}, \mathsf{key})$. As in TT systems, the key generation uses master secret key to sample user private keys sk_j for any given index $j \in [n]$, while the

[5] Although one could ask the tracer to output a set of index-identity pairs instead of only identities, this seems unnecessary as the user index can always be embedded in its identity.

PLBE encryption algorithm uses the public key to encrypt messages. The index-encryption algorithm on the other hand uses the index-encryption key to encrypt messages with respect to an index i. Now such a ciphertext can be decrypted using sk_j only if $j \geq i$, thus one could consider such ciphertexts as encrypting messages under the comparison predicate '$\geq i$'. The security requirements are defined in an 'FE-like' way; that is, if an adversary does not have a key for index i, then index-encryption of any message m to index i should be indistinguishable from index-encryption of m to index $i+1$. Additionally, public key encryptions of any message m should also be indistinguishable from index-encryptions of same message for index 1 (even if adversary is given all keys). And finally, index-encryptions to index $n+1$ should completely hide any information about the encrypted message.

BSW showed that the PLBE framework could be very useful for building TT systems. At a very high level, their main idea was to use the index-encryption functionality to build the tracing algorithm. The tracing algorithm, given access to a decoding algorithm D, estimates the successful decryption probability of index-encryptions to different indices in 1 to $n+1$ when decrypted using algorithm D. If it finds an index i such that the probability estimates corresponding to index-encryptions to i and $i+1$ are noticeably far, then the tracing algorithm includes index i to the set of traitors. In prior works [BSW06, GKW18], it was shown that such a transformation preserves IND-CPA security as well as guarantees secure and correct tracing.

An important aspect of the tracing schema described above is that during tracing the algorithm essentially runs a brute force search over set of user indices $\{1, 2, \ldots, n\}$ to look for traitors. This turns out to be problematic if we want to embed polynomial length identities in the secret keys. Because now the search space for traitors is exponential which turns the above brute force search mechanism rather useless. Thus it is not very clear whether the PLBE framework is an accurate abstraction for 'embedded identity' TT.

In this work, our intuition is to extend the PLBE framework such that it becomes more conducive for implementing the embedded identity tracing functionality in TT systems. Hence, we propose a new PLBE framework called embedded-identity PLBE. As in PLBE, an EIPLBE scheme consists of a setup, key generation, encryption, decryption and special-encryption algorithm. (Here special-encryption algorithm is meant to replace/extend the index-encryption algorithm provided in general PLBE schemes.) Semantically, the differences between PLBE and EIPLBE are as follows. In EIPLBE, the user keys are associated with an index-identity pair (j, id). And, special-encryptions are associated with a index-position-bit tuple (i, ℓ, b), where position is a symbol in $[\kappa] \cup \{\perp\}$. The special-encryption ciphertexts can further be categorized into two types:

($\ell = \perp$) In this case the special-encryption ciphertext for index-position-bit tuple $(i, \ell = \perp, b)$ behaves identical to a PLBE index-encryption to index i. That is, such ciphertexts can be decrypted using $\mathsf{sk}_{j,\mathsf{id}}$ as long as $j \geq i$.

($\ell \neq \perp$) In this case the ciphertext can be decrypted using $\mathsf{sk}_{j,\mathsf{id}}$ as long as either $j \geq i+1$ or $(j, \mathsf{id}_\ell) = (i, 1-b)$. In words, these ciphertexts behave same as a PLBE index-encryption to index i, except decryption by the users corresponding to index-identity pair (i, id) is also disallowed if ℓ^{th} bit of their id matches bit value b.

In short, the special-encryption algorithm (when compared with PLBE index-encryption) provides an additional capability of disabling decryption ability of users depending upon a single bit of their identity. The central idea behind introducing this new capability is that it facilitates a simple mechanism for tracing the identity bit-by-bit. The tracing algorithm runs as a two-step process where the first phase is exactly same as in the PLBE to TT transformation which is to trace the indices of corrupt users. This can be executed as before by using the PLBE functionality of disabling each index one-by-one, that is estimate successful decryption probability of encryptions to indices in 1 to $n+1$ while keeping position variable $\ell = \perp$. This is followed by the core *identity tracing phase* in which the tracing algorithm performs a sub-search on each user index i where it noticed a gap in first phase. Basically the sub-search corresponds to picking a target index obtained during first phase, and then sequentially testing whether the ℓ^{th} bit in the corrupted identity is zero or one for all positions $\ell \in [\kappa]$. And, this is where the above additional disabling capability is used.

Next we discuss the expanded set of security properties required from EIPLBE. More details on the above transformation are provided afterwards.

normal-hiding. Standard encryptions are indistinguishable from special-encryptions to $(1, \perp, 0)$.

index-hiding. Special-encryptions to $(i, \perp, 0)$ are indistinguishable from special-encryptions to $(i+1, \perp, 0)$ if an adversary has no secret key for index i.

lower-ID-hiding. Special-encryptions to $(i, \perp, 0)$ are indistinguishable from special-encryptions to (i, ℓ, b) if an adversary has no secret key for index i and identity id such that $\mathsf{id}_\ell = b$.

upper-ID-hiding. Special-encryptions to $(i+1, \perp, 0)$ are indistinguishable from special-encryptions to (i, ℓ, b) if an adversary has no secret key for index i and identity id such that $\mathsf{id}_\ell = 1 - b$.

message-hiding. Special-encryptions to $(n+1, \perp, 0)$ hide the message encrypted.

Building Indexed EITT from EIPLBE. The setup, key generation, encryption and decryption algorithms for the tracing scheme are same as that for the underlying EIPLBE scheme. Let us now look at how to trace identities from the pirate decoding device. As mentioned before, the tracing proceeds in two phases — (1) index tracing, followed by (2) identity tracing. The idea is to first trace the set of indices of the corrupted users, say $S_{\text{index}} \subseteq [n]$, and then in the second phase for each index $i \in S_{\text{index}}$, the tracer will (bit-by-bit) extract the corresponding identity corrupted. Formally, the tracing proceeds as follows

Phase 1. For $i \in [n+1]$, do the following:

A. Compute polynomially many special-encryptions to index-position-bit $(i, \perp, 0)$.

B. Run decoder D on each ciphertext individually to test whether it decrypts correctly or not. Let $\hat{p}_i$ denote the fraction of successful decryptions.

Let S_{index} denote the set of indices i of such that $\hat{p}_i$ and $\hat{p}_{i+1}$ are noticeably far.

Phase 2. Next, for each $i \in S_{\text{index}}$ and $\ell \in [\kappa]$, do the following:

A. Compute polynomially many special-encryptions to index-position-bit $(i, \ell, 0)$.

B. Run decoder D on each ciphertext individually to test whether it decrypts correctly or not. Let $\hat{q}_{i,\ell}$ denote the fraction of successful decryptions.

Output Phase. Finally, for each $i \in S_{\text{index}}$, it sets the associated traced identity id as follows. For each $\ell \in [\kappa]$, if $\hat{p}_i$ and $\hat{q}_{i,\ell}$ are noticeably far, then set ℓ^{th} bit of id to be 0, else sets it to be 1.

Let us now see why this tracing algorithm works. In the above procedure, the first phase (index tracing) is identical to the PLBE-based tracing algorithm. Thus, by a similar argument it follows that if $i \in S_{\text{index}}$, then it suggests that the decoder D was created using a key corresponding to index-identity pair (i, id) for some identity id. (This part of the argument only relies on normal-hiding, index-hiding and message-hiding security properties.)

The more interesting component of the tracing algorithm is the *identity tracing* phase (i.e., phase 2). The idea here is to selectively disable the decryption ability of users for a fixed index if a particular bit in their identities is 0. Recall that an adversary can not distinguish between special-encryptions to tuple $(i, \perp, 0)$ and $(i, \ell, 0)$ as long as it does not have any secret key for (i, id) such that $\mathsf{id}_\ell = 0$. This follows from 'lower-ID-hiding' property. Similarly, an adversary can not distinguish between special-encryptions to tuple $(i+1, \perp, 0)$ and $(i, \ell, 0)$ as long as it does not have any secret key for (i, id) such that $\mathsf{id}_\ell = 1$. This follows from 'upper-ID-hiding' property. Now whenever $i \in S_{\text{index}}$ we know that $\hat{p}_i$ and $\hat{p}_{i+1}$ are noticeably far. Also, recall that in indexed EITT tracing definition the adversary is allowed to key query for *at most* one identity per index. Therefore, the estimate $\hat{q}_{i,\ell}$ will either be close to $\hat{p}_i$ or to $\hat{p}_{i+1}$, as otherwise one of upper/lower-ID-hiding properties will be violated. Combining all these observations, we can prove correctness/security of the above tracing algorithm.

Next, we move to standard assumption constructions for EIPLBE schemes.

2.3 Building EIPLBE from standard assumptions

In this section, we provide three different pathways for securely realizing embedded-identity private linear broadcast encryption systems under standard assumptions. Our first instantiation is based only on general public key encryption, and is provided to serve as a baseline benchmark for comparing efficiency of other schemes. Our second instantiation is based on Bilinear maps, and provides a quadratic improvement over the PKE-based scheme. And finally, our third and last instantiation is based on learning with errors, and it leads to extremely efficient system parameters. See Table 1 for concrete efficiency comparison. Below we discuss these three approaches in greater detail highlighting the main challenges and contributions. Throughout this section, we use n to denote the maximum number of indices and κ to be the length of identities.

2.3.1 EIPLBE via Public Key Encryption

We first present a EIPLBE scheme based on any PKE scheme. In this scheme, the size of the ciphertexts grows linearly with the maximum number of indices n and the length of identities κ. To understand the intuition behind the PKE based EIPLBE construction, let us recall the folklore PLBE construction based on PKE.

PKE-based PLBE scheme. The setup algorithm chooses n PKE keys $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]}$. A secret key for index i is simply sk_i. Standard encryption of message m consists of n ciphertexts, where the i^{th} ciphertext is an encryption of m under public key pk_i. A special-encryption of m for index i^* consists of n ciphertexts; the first i^* ciphertexts are encryptions of a special symbol $\perp$ (under the respective public keys) while the remaining are encryptions of m (under the respective public keys). In summary, the ciphertext consists of n independent and disjoint components, where each component contains one PKE sub-ciphertext. Thus a user can perform decryption by only looking at its dedicated PKE component in the ciphertext. And security follows directly from PKE security since all the PKE sub-ciphertexts are independently created.

Extending this to EIPLBE. Let us now look at how to extend the simple PLBE scheme described above to embed identities as well. Once again, we will have n different strands, and each strand will have 2κ slots. (Here we perform a PKE setup for each slot in each strand.) A secret key for index i and identity id can unlock κ out of the 2κ slots of the i^{th} strand, and using these κ unlocked components, the decryption algorithm tries to reconstruct a message. In particular, the secret key (i, id) can unlock each of the $\{(\ell, \mathsf{id}_\ell)\}_\ell$ slots. This is executed by giving out the PKE secret keys associated with these slots.

To encrypt a message m, one first creates n copies of the message, and secret shares each copy (independently) into κ shares. Let $\{r_{i,\ell}\}_{\ell\in[\kappa]}$ denote the κ shares of the i^{th} copy. In the i^{th} strand, the $(\ell, 0)$ and $(\ell, 1)$ slots encrypt the same message $r_{i,\ell}$. (Here the per-slot per-strand encryption is performed under the corresponding PKE public key.) As a result, a secret key for index i and identity id can recover all the $\{r_{i,\ell}\}_\ell$ components, and therefore the decryption algorithm can reconstruct the message m.

A special-encryption for index-position-bit tuple (i^*, ℓ^*, b^*) is more involved. In the first $i^* - 1$ strands, it has no information about the message m (it secret shares $\perp$ and puts the shares in the 2κ slots). For all $i > i^*$, the i^{th} strand is constructed just as in the standard encryption (secret share message m into κ shares, and put the ℓ^{th} share in the slots $(\ell, 0)$ and $(\ell, 1)$). The i^* strand is set up in a more subtle way; here, the encryption algorithm again breaks down m into κ shares $\{r_{i^*,\ell}\}_\ell$. It puts $r_{i^*,\ell}$ in slots $(\ell, 0)$ and $(\ell, 1)$ for all ℓ except ℓ^*. In slot (ℓ^*, b^*) it puts $\perp$, and in slot $(\ell^*, 1 - b^*)$ it puts r_{i^*,ℓ^*}. As a result, a secret key for index i^* and identity id such that $\mathsf{id}_{\ell^*} = b^*$ cannot recover r_{i^*,ℓ^*}, and therefore cannot reconstruct the message.

The security properties follow directly from IND-CPA security of the underlying PKE scheme. Consider, for instance, the index hiding property (special-encryption to $(i, \perp, 0)$ is indistinguishable from special-encryption to $(i + 1, \perp, 0)$ if an adversary has no secret keys for index i). The only difference between these two special-encryptions is the ciphertext components in the $\{(\ell, 0), (\ell, 1)\}_\ell$ slots of i^{th} strand. But since the adversary gets no secret keys for index i, it does not have any secret keys to unlock these strand i slots, and hence the index-hiding property holds. The other security properties also follow in a similar manner, except while arguing that the scheme satisfies upper-ID-hiding security we have to additionally use the fact that the message is randomly and independently split in each strand.

The ciphertext size in the above construction grows linearly with both n and κ. Next, we will see how to achieve better parameters using bilinear maps.

2.3.2 EIPLBE via Bilinear maps

When studying EIPLBE, a natural question to ask is whether it can realized generically from standard PLBE schemes. Since we already have bilinear-map based PLBE constructions [BSW06, BW06] in which the size of ciphertext grows linearly with $\sqrt{n}$, thus a generic transformation from PLBE to EIPLBE could probably lead to a bilinear-map solution for EIPLBE with similarly efficiency. Here we consider a very natural such transformation from PLBE to EIPLBE and discuss the challenges faced in executing this approach in a black-box way. Starting with this black-box approach we dig deeper into the existing PLBE schemes and extend them directly to a EIPLBE scheme. More details follow.

Why generic transformation from PLBE to EIPLBE does not work? Let us first describe a simple candidate EIPLBE scheme based on PLBE. The starting point for this transformation is the PKE-based construction described previously. The intuition is to replace each 'strand' sequence in the PKE-based solution with a single PLBE instantiation while keeping the slot structure intact. That is, during setup the algorithm now runs PLBE setup 2κ times — once for each slot in $\{(\ell, b)\}_{\ell,b}$. The public/master secret key consists of the 2κ public/master secret keys $\left\{\mathsf{pk}_{\ell,b}, \mathsf{msk}_{\ell,b}\right\}_{\ell,b}$, one from each slot $(\ell, b) \in [\kappa] \times \{0,1\}$. And, a secret key for index-identity pair (i, id) consists of κ PLBE secret keys, where the ℓ^{th} key component is a secret key for index i in the (ℓ, id_ℓ) slot (that is, $\mathsf{sk} = \{\mathsf{sk}_\ell\}_\ell$ where $\mathsf{sk}_\ell \leftarrow \mathsf{KeyGen}(\mathsf{msk}_{\ell,\mathsf{id}_\ell}, i)$). Next, let us look at encryption. A ciphertext consists of 2κ PLBE ciphertexts $\{\mathsf{ct}_{\ell,b}\}_{\ell,b}$. The (standard) encryption algorithm splits message m into κ shares $\{r_\ell\}_\ell$, and then encrypts r_ℓ under the public keys for both $(\ell, 0)$ and $(\ell, 1)$ slots, independently. The special-encryption algorithm on the other hand works as follows — to encrypt m for index-position-bit tuple (i^*, ℓ^*, b^*), the algorithm as before splits m into κ shares $\{r_\ell\}_\ell$, and then computes all but the (ℓ^*, b^*)-slot of the ciphertext as a PLBE index-encryption (of the corresponding share) for index i^*. And, the last remaining ciphertext component (if any[6]) is a PLBE index-encryption (of the corresponding share) for index '$i^* + 1$'. Now decryption can be quite naturally defined. Let us next try to analyze its security.

A careful inspection of the above scheme shows that it satisfies all requisite security properties except one which is upper-ID-hiding security.[7] Recall that upper-ID-hiding security requires that special-encryption to $(i+1, \perp, 0)$ must be indistinguishable from special-encryption to (i, ℓ, b) if the adversary doed not get any secret key for (i, id) such that $\mathsf{id}_\ell = 1 - b$. Suppose an adversary has two secret keys $\mathsf{sk}_{i,\mathsf{id}}$ and $\mathsf{sk}_{i+1,\mathsf{id}}$, for some identity id such that $\mathsf{id}_\ell = b$. Consider a new secret key $\widetilde{\mathsf{sk}}$ which is equal to $\mathsf{sk}_{i,\mathsf{id}}$, except that the ℓ^{th} component is set to be the ℓ^{th} component of $\mathsf{sk}_{i+1,\mathsf{id}}$. It turns out that this hybrid key $\widetilde{\mathsf{sk}}$ can decrypt a special-encryption for (i, ℓ, b) but not for $(i+1, \perp, 0)$, even though both key queries for index-identity pairs (i, id) and $(i+1, \mathsf{id})$ are permissible as per upper-ID-hiding security game.

As exhibited by the above attack, the main issue with the above (broken) candidate is that there is no mechanism to tie together the different components of a particular secret key. Thus such key mixing attacks, which allow rendering hybrid keys such as $\widetilde{\mathsf{sk}}$ in the aforementioned attack, are unavoidable. In order to prevent such attacks, we dive into the existing PLBE constructions with the goal of exploiting the underlying algebraic structure for linking together the individual PLBE secret keys coming from different subsystems.

Our intuition and fixing [BW06]. Our starting point is the trace and revoke (broadcast) scheme by Boneh and Waters (BW) [BW06]. We start by presenting a simplified version of the BW PLBE scheme, and then use that as a building block to build our EIPLBE scheme. Along the way we uncover a crucial bug in

[6]If $\ell^* = \perp$, then all ciphertext slots have already been filled.

[7]Actually there is a pretty simple (related) attack to break the false tracing guarantee if one uses this transformation to build an indexed EITT scheme from standard PLBE. Here we only focus on breaking upper-ID-hiding security..

the security proof provided by BW that renders their theorem as stated incorrect. In this work, we fix the BW security proof while building our EIPLBE scheme, thereby restoring the bilinear map based TT (also trace and revoke) schemes to their original glory.

Revisiting BW tracing scheme. Let p, q be primes, $\mathbb{G}, \mathbb{G}_T$ groups of order $N = p \cdot q$ with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and let $\mathbb{G}_p, \mathbb{G}_q$ denote the subgroups of $\mathbb{G}$ of orders p and q respectively. In the BW tracing scheme for n parties, any index $i \in [n]$ is represented as a pair $(i_1, i_2) \in [\sqrt{n}] \times [\sqrt{n}]$; secret keys and special-encryptions are for pairs $(x, y) \in [\sqrt{n}] \times [\sqrt{n}]$. We say that $(x_1, y_1) \prec (x_2, y_2)$ if either $x_1 < x_2$ or ($x_1 = x_2$ and $y_1 < y_2$).

The setup algorithm chooses generator $g \leftarrow \mathbb{G}$ and $g_q \leftarrow \mathbb{G}_q$, scalars $\alpha_x, r_x, c_x \leftarrow \mathbb{Z}_N$ for each $x \in [\sqrt{n}]$ and sets $E_x = g^{r_x}$, $G_x = e(g, g)^{\alpha_x}$ and $H_x = g^{c_x}$. It chooses $\beta \leftarrow \mathbb{Z}_N$, sets $E_q = g_q^{\beta}$, $E_{q,x} = g_q^{\beta r_x}$ and $G_{q,x} = e(g_q, g_q)^{\beta\alpha_x}$. The public key consists of $\{E_x, G_x, E_q, E_{q,x}, G_{q,x}, H_x\}_x$ (together with some additional components); the master secret key consists of $\{\alpha_x, r_x, c_x\}_x$, and the tracing key is the public key itself. A secret key for index (x, y) is set to be $g^{\alpha_x + r_x c_y}$. Special-encryption of message m for index (x^*, y^*) has $4\sqrt{n}$ components $\{R_i, A_i, B_i, C_i\}_{i\in[\sqrt{n}]}$. It chooses $s_x \leftarrow \mathbb{Z}_N$ for each $x \in [\sqrt{n}]$, $t \leftarrow \mathbb{Z}_N$. For $x > x^*$, it sets $R_x = E_{q,x}^{s_x} = g_q^{\beta r_x s_x}$, $A_x = E_q^{s_x t} = g_q^{\beta s_x t}$ and $B_x = m \cdot G_{q,x}^{s_x t} = m \cdot e(g_q, g_q)^{\beta\alpha_x s_x t}$. For $x = x^*$, it sets $R_x = E_x^{s_x} = g^{r_x s_x}$, $A_x = g^{s_x t}$ and $B_x = m \cdot G_x^{s_x t} = m \cdot e(g, g)^{\alpha_x s_x t}$. For $x < x^*$, R_x, A_x, B_x are random group elements. Next, it sets C_y as follows. For $y > y^*$, it sets $C_y = H_y^t = g^{c_y t}$; else it sets $C_y = g^{c_y t} \cdot h_p$, where h_p is a group element in $\mathbb{G}_p$, derived from the public parameters.

For correctness, let $K = g^{\alpha_x + r_x c_y}$ be a key for (x, y), $\mathsf{ct} = \{R_i, A_i, B_i, C_i\}_i$ an encryption of m for (x', y'), where $(x', y') \prec (x, y)$. Consider the terms (R_x, A_x, B_x, C_x). If $x > x'$, then $R_x = g_q^{\beta r_x s_x}$, $A_x = g_q^{\beta s_x t}$, $B_x = e(g_q, g_q)^{\beta\alpha_x r_x s_x t}$ for some β, s_x, t. On pairing R_x with C_y, one obtains $\Gamma_1 = e(g_q, g_q)^{\beta r_x s_x t c_y}$. Here, note that it does not matter whether $y < y'$ or not, because pairing an element in $\mathbb{G}_p$ with an element in $\mathbb{G}_q$ results in identity. Next, pairing A_x with the secret key K results in $\Gamma_2 = e(g_q, g_q)^{\beta r_x s_x c_y t + \alpha_x r_x s_x}$. Finally, note that $B_x \cdot \Gamma_1/\Gamma_2 = m$. If $x = x'$ but $y > y'$, then pairing A_x and K results in $\Gamma_2 = e(g, g)^{r_x s_x c_y t + \alpha_x r_x s_x}$, and pairing R_x and C_y results in $e(g, g)^{r_x s_x c_y t}$. Therefore $B_x \cdot \Gamma_1/\Gamma_2$ outputs m.

The main intuition behind the index-hiding security proof is that if an adversary does not have a secret key for index $i = (x, y)$, then the h_p term multiplied to C_y component can be undetectably added or removed. In the actual scheme, the public parameters and the ciphertext includes some additional terms for security purposes. Here we removed them for simplicity of exposition. Next, let us look at how to extend BW for building an EIPLBE scheme.

Our EIPLBE scheme based on Bilinear Maps. Our EIPLBE scheme, at a very high level, is inspired by the 2κ-subsystems idea (described in the attempted generic transformation from PLBE to EIPLBE) applied to the BW scheme. However, we will ensure that the adversary cannot mix-and-match different secret keys. Consider 2κ different subsystems of the BW scheme, where all the subsystems share the same $\{\alpha_x, r_x\}_{x\in[\sqrt{n}]}$ values, but each subsystem has its own $\{c_y\}_{y\in[\sqrt{n}]}$ values. So, the public key has $\{E_x, G_x, E_q, E_{q,x}, G_{q,x}\}_x$ (together with some additional components) as in the BW scheme, but instead of $\{H_y\}_{y\in[\sqrt{n}]}$, it now has $\{H_{y,\ell,b}\}_{y\in[\sqrt{n}],\ell\in[\kappa],b\in\{0,1\}}$, where the setup algorithm chooses $\{c_{y,\ell,b}\}_{y\in[\sqrt{n}]}$ values for the (ℓ, b) subsystem and sets $H_{y,\ell,b} = g^{c_{y,\ell,b}}$. The secret key for index $i = (x, y)$ and identity id consists of just one component. The key generation algorithm combines the appropriate $c_{y,\ell,b}$ elements (depending on id) and multiplies with r_x. Let $\gamma_{x,y} = r_x \cdot (\sum_\ell c_{y,\ell,\mathsf{id}_\ell})$. The key generation algorithm outputs $g^{\alpha_x + \gamma_{x,y}}$ as the secret key. Note that unlike the PLBE to EIPLBE transformation, here the components from one key cannot be mixed with the components of another key to produce a hybrid key. An alternate view of the secret key is that it is the BW key, but with c_y value being different for each identity (for identity id, $c_y = \sum_\ell c_{y,\ell,\mathsf{id}_\ell}$).

In the ciphertext/special-ciphertext, we have the $\{R_x, A_x, B_x\}_{x\in[\sqrt{n}]}$ components as in the BW scheme. However, instead of $\{C_y\}_{y\in[\sqrt{n}]}$, we now have 2κ such sets of components. During decryption, one must first combine the $C_{y,\ell,b}$ components depending on the identity id to obtain a term C_y, which is then used to carry out BW-like decryption. We will now present the scheme in more detail.

The setup algorithm chooses $\{c_{y,\ell,b}\}_{y\in[\sqrt{n}],\ell\in[\kappa],b\in\{0,1\}}$. It sets $H_{y,\ell,b} = g^{c_{y,\ell,b}}$ for each $(y, \ell, b) \in [\sqrt{n}] \times [\kappa] \times \{0, 1\}$, and the public key is $\left\{E_x, G_x, E_q, E_{q,x}, G_{q,x}, \{H_{x,\ell,b}\}_{\ell,b}\right\}_x$, where the $E_x, G_x, E_q, E_{q,x}, G_{q,x}$ terms are computed as in the BW scheme (outlined above). To compute a secret key for index (x, y) and identity id, the key generation algorithm computes $z = \alpha_x + r_x \cdot (\sum_i c_{y,i,\mathsf{id}_i})$ and outputs g^z as the secret key. Finally, the special-encryption of m for index (x^*, y^*), position ℓ^* and bit b^* is computed as follows: for each $x \in [\sqrt{n}]$, the encryption algorithm computes $\{R_x, A_x, B_x\}$ as in the BW scheme. In addition to these components, it computes $\{C_{y,\ell,b}\}$ components for each $y \in [\sqrt{n}], \ell \in [\kappa]$ and $b \in \{0, 1\}$ as follows: if $(y > y^*)$ or $(y = y^*$ and $(\ell, b) \neq (\ell^*, b^*))^t$, then $C_{y,\ell,b} = H^t_{y,l,b}$, else $C_{y,\ell,b} = H^t_{y,l,b} \cdot h_p$, where h_p is some element in $\mathbb{G}_p$ computed using the public parameters.

Suppose K is a key for index (x, y) and identity id, and $\left\{R_x, A_x, B_x, \{C_{x,l,b}\}_{l,b}\right\}_x$ is an encryption of m for $((x^*, y^*), \ell^*, b^*)$. Decryption works as follows: first, compute $C_y = \prod_l C_{y,l,\mathsf{id}_l}$; next, pair C_y and A_x to compute Γ_1, pair K and R_x to compute Γ_2, and output $B_x \cdot \Gamma_1/\Gamma_2$ as the decryption.

The full scheme and security proof is discussed in detail later in Section 7. Note that in the above outline, the size of ciphertexts grows linearly with $\sqrt{n}$ and κ. In the main body, we optimize the construction such that the size of ciphertexts grows linearly with both $\sqrt{n}$ and $\sqrt{\kappa}$. Finally, we will present a scheme with optimal ciphertext size with only polylogarithmic dependence on n.

2.3.3 EIPLBE via Learning With Errors

In a recent work, Goyal, Koppula, and Waters [GKW18] gave a traitor tracing scheme with compact ciphertexts. Their scheme is based on a new primitive called Mixed Functional Encryption (Mixed FE), which can also be used to build an EIPLBE scheme with optimal parameters. A Mixed FE scheme for a function class $\mathcal{F}$ can be seen as an extension of a secret key FE scheme for $\mathcal{F}$. It has a setup, key generation, encryption and decryption algorithm (as in a secret key FE scheme). In addition, it also has a public encryption algorithm. For the PLBE and EIPLBE schemes, it helps to have keys associated with messages and ciphertexts with functions. The setup algorithm chooses a public key pk and a master secret key msk. The master secret key can be used to generate a secret key for any message m, and can also be used to encrypt any function f. A key for message m can decrypt an encryption of function f if $f(m) = 1$. In addition, the public-encryption algorithm can also generate ciphertexts; it only takes as input the public key pk, and outputs a ciphertext that 'looks like' a secret-key encryption of the 'all-accepting function'. For security, GKW require bounded query FE security, together with the public/secret key mode indistinguishability.

The work of [GKW18] showed a construction of Mixed FE for log-depth circuits. A recent work by Chen et al. [CVW+18] showed three different constructions for the same. To construct PLBE, [GKW18] combined a 1-bounded Mixed FE scheme with an ABE scheme. The PLBE encryption of a message m is simply an ABE encryption of m for attribute x being a public-mode Mixed FE encryption. The special-encryption of m for index i^* is again an ABE encryption of m, but with attribute x being a secret-key Mixed FE encryption of the $(> i^*)$ function. Finally, to compute a secret key for index i, the key generation algorithm first computes a Mixed FE key k for the message i, and then computes an ABE key for a Mixed FE decryption circuit that has k hardwired, takes a Mixed FE ciphertext ct as input and outputs Mixed FE decryption of ct using k. Note that for this transformation, it suffices to only have a Mixed FE scheme that allows the comparison functionality.

Fortunately (for us), [GKW18] (and later [CVW+18]) showed Mixed FE for a much richer class of functions (log-depth circuits), and this will be useful for our construction. Our EIPLBE scheme will also follow the Mixed FE+ABE approach (which is referred to as Mixed FE with messages in [CVW+18]). Instead of the comparison function, the Mixed FE ciphertexts in our scheme will be for more expressive functions. In particular, it suffices to have a Mixed FE scheme where the functions are parameterized by (y^*, ℓ^*, b^*), and it checks if input (y, id) either satisfies $y > y^*$, or $y = y^*$ and $\mathsf{id}_\ell \neq b^*$. Since such simple functions can be implemented in log-depth, we can use the ABE+Mixed FE approach for building EIPLBE as well.

2.4 Indexed Embedded-Identity TT to Bounded Embedded-Identity TT

In this part, we discuss our transformation from a tracing scheme with indexed key generation to one where there is no index involved, but the correct trace guarantee holds only if total number of keys is less than an apriori set bound. For technical reasons we require the bounded EITT system to provide a stronger false tracing guarantee, which states there should be no false trace even if the adversary obtains an unbounded (but polynomial) number of keys. Looking ahead, this property will be crucial for the transformation from bounded EITT to its unbounded counterpart.

The high-level idea is to have λ different strands, and in each strand, we have a separate indexed-system with a large enough index bound (that depends on the bound on number of keys n). When generating a key, we choose λ random indices (within the index bound) and generate λ different keys for the same identity in the different strands using the respective randomly chosen indices. Now, we will set the index bound to be n^2, and as a result, at least one strand has all distinct indices (with overwhelming probability). To (special-)encrypt a message, we secret-share the message in the λ different strands, and encrypt them separately. This approach satisfies the correct-trace guarantee, but does not satisfy the false-trace guarantee. In particular, note that the false-trace guarantee should hold even if the number of key queries is more than the query bound. This means the underlying indexed scheme should not report a false trace even if there are multiple identities for a index, which is a strictly stronger false-trace guarantee for the underlying system (and our system does not satisfy it).

There is an elegant fix to this issue. Instead of generating keys for the queried identity id, the key generation algorithm now generates a signature on id, and generates keys for (id, σ). This fixes the false-trace issue. Even if an adversary queries for many secret keys, if it is able to produce a decoding box that can implicate a honest user, then that means this box is able to forge signatures, thereby breaking the signature scheme's security. We describe the scheme a little more formally now.

To build a tracing scheme with bound n, the setup algorithm chooses λ different public/secret/tracing keys for the indexed scheme with index bound set to be n^2. The setup algorithm also chooses a signature key/verification key. It sets the λ different public keys and the verification key to be the new public key, and similarly the master secret key has the λ different master secret keys and the signature key. Encryption of a message m works as follows: the encryption algorithm chooses λ shares of the message, and then encrypts the i^{th} share under the i^{th} public key. To compute a secret key for identity id, the key generation algorithm first chooses λ different indices $j_1, \ldots, j_\lambda$. It then computes a signature σ on id, and generates a key for (id, σ) using each of the λ master secret keys with the corresponding indices. The tracing algorithm uses the underlying indexed scheme's trace algorithm to obtain a set of (id, σ) tuples. It then checks if σ is a valid signature on id; if so, it outputs id as a traitor.

Now, suppose an adversary queries for $t(< n)$ secret keys, and outputs a decoding box D. Let $j_{i,k}$ denote the k^{th} index chosen for the i^{th} secret key. With high probability, there exists an index $k^* \in [\lambda]$ such that the set of indices $\{j_{1,k^*}, j_{2,k^*}, \ldots, j_{t,k^*}\}$ are all distinct. As a result, using the correct-tracing guarantee of the underlying tracing scheme for the k^* strand, we can extract at least one tuple (id, σ).

Next, we need to argue the false trace guarantee. This follows mainly from the security of the signature scheme. Suppose an adversary receives a set of keys corresponding to an identity set $\mathcal{I}$, and outputs a decoding box D. If trace outputs an identity $\mathsf{id} \notin \mathcal{I}$, then this means the sub-trace algorithm output a tuple (id, σ) such that σ is a valid signature on id. As a result, σ is a forgery on message id (because the adversary did not query for a key corresponding to id).

2.5 Bounded Embedded-Identity TT to Unbounded Embedded-Identity TT

The final component is to transform a tracing system for bounded keys to one with no bound on the number of keys issued. For this transformation to be efficient, it is essential that the underlying bounded EITT scheme to have ciphertexts with polylogarithmic dependence on the key bound n. The reason is that our core idea is to have λ (bounded) EITT systems running in parallel, where the i^{th} system runs the bounded tracing scheme with bound $n_i = 2^i$, and if the ciphertext size does not scale polylogarithmically with the bound n_i, then this transformation would not work.[8]

More formally, the setup algorithm runs the bounded system's setup λ times, the i^{th} iteration run with bound $n_i = 2^i$. It sets the public key (resp. master secret key and the tracing key) to be the λ public keys (resp. the λ different master secret keys and the tracing keys). The encryption algorithm secret shares the message into λ shares, and encrypts the i^{th} share using the i^{th} public key. The key generation algorithm computes λ different secret keys. Finally, the tracing algorithm runs the bounded system's trace algorithm, one by one, until it finds a traitor. First, note that since the adversary is polynomially bounded, if it queries for t keys, then there exists some i^* such that $t \leq 2^{i^*} < 2t$. As a result, the trace is guaranteed to find a traitor in the i^{*th} system, and hence it runs in time $\mathsf{poly}(2^{i^*}) = \mathsf{poly}(t)$. Second, since every underlying bounded system's false trace guarantee holds even if the adversary queries for more keys than permitted, thus none of the premature sub-traces result in a false trace. At a very high level, the central observation here that allows us in avoiding the need for adaptive security is that: while tracing we simply perform the "tighest" fit search for finding the smallest polynomial bound on keys corrupted and then carry out the tracing procedure rather than tracing on an exponential sized space directly. Similar techniques of combining different *bounded adversary* instances, and invoking the security of the instance with *just high enough* security were used previously in [BHJ+13, DS15].

2.6 Comparing Techniques

We conclude by giving some further comparisons between the techniques we introduce and those from the earlier work of NWZ [NWZ16]. The closest point for comparisons are the techniques they use to trace an identity of arbitrary size κ while keeping ciphertexts possibly smaller than κ bits. (We modify their variable names to more closely match ours.) Here they introduce a sub-primitive called private *block* linear broadcast encryption (PBLBE) which can be used as follows. A private key for identity $\mathsf{id} = (\mathsf{id}_1, \mathsf{id}_2 \cdots, \mathsf{id}_\kappa)$ will be associated with a randomly chosen tag s from an exponential sized space. It is then organized into i blocks where each block is associated with the pair (s, id_j) which is embedded by the value $2s + \mathsf{id}_j$. Given a decoding algorithm D the tracing algorithm will perform a search procedure on *each* individual block to recover the set of corrupted tag/identity bit values on each one. The process will essentially perform a search on the j-th block values while leaving all blocks $k \neq j$ alone. At the end, the tracing process will look for a tag s^* that is present in all the individual block searches and use that to reconstruct the traitor identity. An analysis is needed to show that such a tag exists so that one is not just stuck with fragments of many different identities.

At a high level our indexed EITT two part structure (consisting of an index i and identity id) is similar to the two part structure of [NWZ16] consisting of a tag s along with the identity. However, there exists some important differences that are closely linked to our goal of realizing embedded traitor tracing from standard assumptions.

- First, our tracing procedure searches in a qualitatively different manner where it first performs a search across the index space (without regard) to identity bits and only when an index is found does it perform a dive into extracting the identity. This is in contrast to the NWZ approach of performing tag/index search per each identity bit, and then combining the identity bits (corresponding to every unique tag) to reconstruct traitor identities. We believe the current way is simpler and has less tracing overhead. In addition, our indexed EITT interface is intended to be a minimalistic which in general helps for realization from more basic assumptions as opposed to full blown functional encryption.
- We consider indices of small range while the tag spaces of NWZ are exponential size. This enables us to access a wider class of traitor tracing realizations from PKE and bilinear maps. There are no known PLBE schemes for exponentially large identity spaces from these assumptions.
- We achieve our scheme for unbounded identities by amplifying from smaller index sized schemes along with an analysis that finds the "tightest fit". The work of [NWZ16] requires *adaptive* security of the

[8]Due to similar reasons, it is essential that the running time of all algorithms (except possibly the tracing algorithm) grows at most polylogarithmically with n.

underlying primitive. The only known scheme from standard assumptions that can handle exponentially large identity space is the [CVW+18] which builds the core "Mixed FE" component from lockable obfuscation [GKW17, WZ17]. It is notable that the private constrained PRF-based construction of [CVW+18] and the earlier [GKW18] construction of Mixed FE only offer selective security. This suggests that adaptive security may in general be hard to come by and developing techniques to avoid it a worthwhile goal.

Lastly, NWZ also studied the problem in the bounded collusion setting, wherein they provided constructions from regular public-key encryption (instead of full blown FE) where the size of ciphertexts and parameters grew at least linearly in the collusion size. If one sets the collusion size to be the number of users n, then their bounded collusion constructions could be interpreted as collusion-resistant constructions for our indexed EITT notion. However, that approach leads to much less efficient constructions.

3 Preliminaries

Notations. Let PPT denote probabilistic polynomial-time. We denote the set of all positive integers upto n as $[n] := \{1, \dots, n\}$. Throughout this paper, unless specified, all polynomials we consider are positive polynomials. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each component is drawn independently from the distribution $\mathcal{D}$.

3.1 Bilinear Groups and Assumptions

In this work, we will use composite order bilinear groups for our main construction. Let Gen be a PPT algorithm that takes as input a security parameter λ and outputs a tuple $(p, q, N = pq, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot))$ where p, q are two distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order N, and e is an efficiently computable function mapping two group elements of $\mathbb{G}$ to a group element in $\mathbb{G}_T$ and satisfying the following properties:

- Bilinearity: $\forall g \in \mathbb{G}, a, b \in \mathbb{Z}_N, e(g^a, g^b) = e(g,g)^{ab}$,
- Non-Degeneracy: $e(g,g) \neq 1_{\mathbb{G}_T}$ for $g \neq 1_{\mathbb{G}}$, where $1_{\mathbb{G}}$ and $1_{\mathbb{G}_T}$ are the identity elements of groups $\mathbb{G}$ and $\mathbb{G}_T$ respectively.

We now recall the assumptions on composite order bilinear groups from Boneh-Waters [BW06]. Here the first three assumptions are exactly what were used in [BW06], and the fourth assumption is new but is required to prove security of the Boneh-Waters scheme also.[9] We will use the notation $\mathbb{G}_p, \mathbb{G}_q$ to denote the respective subgroups of order p and order q of $\mathbb{G}$.

Assumption 1 (Decision (Modified) 3-party Diffie-Hellman Assumption)**.** For every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}\left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_p, g_q, \\ g_p^a, g_p^b, g_p^c, g_p^{b^2}, T_b \end{array}\right) = b : \begin{array}{c} (p, q, N = pq, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda); \\ g_p \leftarrow \mathbb{G}_p;\ g_q \leftarrow \mathbb{G}_q;\ a, b, c, r \leftarrow \mathbb{Z}_p; \\ T_0 = g_p^{abc};\ T_1 = g_p^r;\ b \leftarrow \{0,1\} \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Assumption 2 (Diffie-Hellman Subgroup Decision Assumption)**.** For every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}\left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g, h, \\ g_q^a, h_q^a, g^b g_p^c, h^b, T_b \end{array}\right) = b : \begin{array}{c} (p, q, N = pq, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda); \\ g_p, h_p \leftarrow \mathbb{G}_p;\ g_q, h_q \leftarrow \mathbb{G}_q;\ a, b, c \leftarrow \mathbb{Z}_N; \\ g = g_p g_q;\ h = h_p h_q;\ T_0 \leftarrow \mathbb{G}_q;\ T_1 \leftarrow \mathbb{G};\ b \leftarrow \{0,1\} \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

[9]Recall that the security proof provided in [BW06] is incorrect as discussed in the technical introduction.

Assumption 3 (Bilinear Subgroup Decision Assumption)**.** For every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}\left(N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_p, g_q, e(T_b, g)\right) = b : \begin{array}{c} (p, q, N = pq, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda); \\ g_p \leftarrow \mathbb{G}_p;\ g_q \leftarrow \mathbb{G}_q;\ g \leftarrow \mathbb{G}; \\ T_0 \leftarrow \mathbb{G}_p;\ T_1 \leftarrow \mathbb{G};\ b \leftarrow \{0,1\} \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Assumption 4 (*Relaxed* 3-party Diffie-Hellman Assumption)**.** For every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}\left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_p, g_q, \\ g_q^a, g_p^{\widetilde{a}} g_q^{a^2}, g_p^{\widetilde{c}} g_q^c, T_b \end{array}\right) = b : \begin{array}{c} (p, q, N = pq, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda); \\ g_p \leftarrow \mathbb{G}_p;\ g_q \leftarrow \mathbb{G}_q;\ \widetilde{a}, \widetilde{c} \leftarrow \mathbb{Z}_p;\ a, c \leftarrow \mathbb{Z}_q; \\ T_0 = g_q^{a^2 c};\ T_1 \leftarrow \mathbb{G}_q;\ b \leftarrow \{0,1\} \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

4 Traitor Tracing with Embedded Identities

4.1 Indexed Embedded-Identity Traitor Tracing

In this section, we will present the syntax and definitions for traitor tracing with embedded identities where the number of users is bounded, and the key generation is 'indexed'.

Let $\mathcal{T}$ be a (indexed keygen, public/private)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$. It consists of five algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\text{indx}}) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$: The setup algorithm takes as input the security parameter λ, the 'identity space' parameter κ, index space $[n_{\text{indx}}]$, and outputs a master secret key msk, a public key pk, and a tracing key key.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa, i \in [n_{\text{indx}}]) \to \mathsf{sk}_{i,\mathsf{id}}$: The key generation algorithm takes as input the master secret key, identity $\mathsf{id} \in \{0,1\}^\kappa$ and index $i \in [n_{\text{indx}}]$. It outputs a secret key $\mathsf{sk}_{i,\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$: The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$: The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1) \to T \subseteq \{0,1\}^\kappa$. The trace algorithm has oracle access to a program D, it takes as input key (which is the master secret key msk in a private-key tracing scheme, and the public key pk in a public tracing algorithm), parameter y and two messages m_0, m_1. It outputs a set T of index-identity pairs, where $T \subseteq \{0,1\}^\kappa$.

Correctness. A traitor tracing scheme is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, \kappa, n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, identity $\mathsf{id} \in \{0,1\}^\kappa$ and $i \in [n]$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n); \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, i); \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Efficiency. Let T-s, T-e, T-k, T-d, T-t, S-c, S-k be functions. A (indexed keygen, public/private)-embedded identity tracing scheme is said to be (T-s, T-e, T-k, T-d, T-t, S-c, S-k)- efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\text{indx}})$ is at most $\mathsf{T\text{-}s}(\lambda, \kappa, n_{\text{indx}})$.

- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The number of oracle calls made by $\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1)$ to decoding box D is at most $\mathsf{T\text{-}t}(\lambda, \kappa, n_{\mathrm{indx}}, y)$.
- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa, n_{\mathrm{indx}})$.

Definition 4.1. A traitor tracing scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ is said to have *public tracing* if the tracing algorithm Trace uses the public key.

4.1.1 Security

As in the traditional traitor tracing definitions, we have two security definitions. The first security definition ($\mathsf{IND\text{-}CPA}$ security) states that any PPT adversary should not distinguish between encryptions of different messages. This definition is identical to the INDCPA definition in traditional traitor tracing. The second definition states that if there exists a pirate decoder box, then the tracing algorithm can trace the identity of at least one of the secret keys used to build the decoding box, and there are no 'false-positives'.

Definition 4.2 ($\mathsf{IND\text{-}CPA}$ security)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (indexed keygen, public/private)-embedded identity tracing scheme. This scheme is $\mathsf{IND\text{-}CPA}$ secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \ : \ \begin{array}{c} (1^\kappa, 1^{n_{\mathrm{indx}}}) \leftarrow \mathcal{A}(1^\lambda); (\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{indx}}); \\ b \leftarrow \{0,1\}; (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Definition 4.3 (Secure tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (indexed keygen, public/private)-embedded identity tracing scheme. For any non-negligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Figure 1.

Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^\kappa, 1^{n_{\mathrm{indx}}} \leftarrow \mathcal{A}(1^\lambda)$
- $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{indx}})$
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{key}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$

Each oracle query made by the adversary $\mathcal{A}$ consists of an index-identity pair $(i, \mathsf{id}) \in [n_{\mathrm{indx}}] \times \{0,1\}^\kappa$. Let $S_{\mathcal{ID}}$ the set of identities queried by $\mathcal{A}$. Here, oracle $O(\cdot)$ has msk hardwired and on query (i, id) it outputs $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, i)$ if index i is distinct from all previous queries made by the adversary, otherwise it outputs $\perp$. *In other words, for each index $i \in [n_{\mathrm{indx}}]$, the adversary is allowed to make at most one key query. However, for different indices $i, i' \in [n_{\mathrm{indx}}]$, the identity can be same (that is, (i, id) and (i', id) are valid queries if $i \neq i'$).*

Figure 1: Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}$

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$
  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder}]$.

- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathcal{ID}}$
  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S_{\mathcal{ID}}$
  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathsf{negl}_1(\cdot)$, $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal}\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathsf{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor}\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G}\text{-}\mathsf{D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}_2(\lambda).$$

Remark 4.1. We want to point out that in both IND-CPA and secure tracing games we require the adversary to output the index bound n_{indx} *in unary instead of binary* (i.e., $\mathcal{A}$ outputs $(1^\kappa, 1^{n_{\text{indx}}})$ instead of $(1^\kappa, n_{\text{indx}})$). Now since the running time of the adversary $\mathcal{A}$ is bounded by a polynomial, thus it can only select a polynomially-bounded value for index bound n_{indx}. However, the setup algorithm is given the input n_{indx} *in binary.* This distinction will later be useful in our constructions and security proofs.

5 A New Framework for Embedded-Identity Traitor Tracing

5.1 Embedded-Identity Private Linear Broadcast Encryption

We introduce the notion of embedded-identity private linear broadcast encryption (EIPLBE) as a generalization of private linear broadcast encryption scheme which was introduced by Boneh, Sahai and Waters [BSW06] as a framework for constructing traitor tracing schemes. There are five algorithms in a EIPLBE scheme — $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec}$. The setup algorithm outputs a master secret key and a public key. The key generation algorithm is used to sample private keys for index-identity pairs (j, id). The public key encryption algorithm can be used to encrypt messages, and ciphertexts can be decrypted using any of the private keys via the decryption algorithm. In addition to these algorithms, there is also a special-encryption algorithm SplEnc. This algorithm, which uses the master secret key, can be used to encrypt messages to any index-position-value tuple (i, ℓ, b). A secret key for user (j, id) can decrypt a ciphertext for index-position-value tuple (i, ℓ, b) only if (1) $j \geq i + 1$, or (2) $(i, \ell) = (j, \perp)$ or $(i, \mathsf{id}_\ell) = (j, 1 - b)$.

Belowe we first provide the EIPLBE syntax, and then present the security definitions.

Syntax. A EIPLBE scheme $\mathsf{EIPLBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec})$ for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$ has the following syntax.

$\mathsf{Setup}(1^\lambda, 1^\kappa, n) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm takes as input the security parameter λ, the 'identity space' parameter κ, index space n, and outputs a master secret key msk and a public key pk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa, i \in [n]) \to \mathsf{sk}$. The key generation algorithm takes as input the master secret key, an identity $\mathsf{id} \in \{0,1\}^\kappa$ and index $i \in [n]$. It outputs a secret key sk.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$, and outputs a ciphertext ct.

$\mathsf{SplEnc}(\mathsf{key}, m, (i, \ell, b)) \to \mathsf{ct}$. The special-encryption algorithm takes as input a key key, message $m \in \mathcal{M}_\lambda$, and index-position-value tuple $(i, \ell, b) \in [n+1] \times ([\kappa] \cup \{\perp\}) \times \{0,1\}$, and outputs a ciphertext ct. (Here the scheme is said to be public key EIPLBE scheme if $\mathsf{key} = \mathsf{pk}$. Otherwise, it is said to be private key EIPLBE scheme.)

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

Correctness. A EIPLBE scheme is said to be correct if there exists negligible functions $\mathsf{negl}_1(\cdot)$, $\mathsf{negl}_2(\cdot)$ such that for all $\lambda, \kappa, n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in [n+1]$, $j \in [n]$, $\mathsf{id} \in \{0,1\}^\kappa$, $\ell \in ([\kappa] \cup \{\perp\})$ and $b \in \{0,1\}$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},\mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk},\mathsf{pk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n)\\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk},\mathsf{id}, j)\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\end{array}\right] \geq 1 - \mathsf{negl}_1(\lambda),$$

$$\left(\begin{array}{c}(j \geq i+1)\ \vee\\ (i,\ell) = (j,\perp)\ \vee\\ (i,\mathsf{id}_\ell) = (j, 1-b)\end{array}\right) \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk},\mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk},\mathsf{pk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n)\\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk},\mathsf{id}, j)\\ \mathsf{ct} \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i,\ell,b))\end{array}\right] \geq 1 - \mathsf{negl}_2(\lambda).$$

Efficiency. Let $\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}\widetilde{e}}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$ be functions. A EIPLBE scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}\widetilde{e}}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)- efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa, n)$ is at most $\mathsf{T\text{-}s}(\lambda,\kappa,n)$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda,\kappa,n)$.
- The running time of $\mathsf{SplEnc}(\mathsf{key}, m, (i,\ell,b))$ is at most $\mathsf{T\text{-}\widetilde{e}}(\lambda,\kappa,n)$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk},\mathsf{id}, i)$ is at most $\mathsf{T\text{-}k}(\lambda,\kappa,n)$.
- The running time of $\mathsf{Dec}(\mathsf{sk},\mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda,\kappa,n)$.
- The size of the ciphertexts is at most $\mathsf{S\text{-}c}(\lambda,\kappa,n)$.
- The size of the key is at most $\mathsf{S\text{-}k}(\lambda,\kappa,n)$.

5.1.1 q-query EIPLBE Security

Now we provide the security definitions for EIPLBE as a generalization of the PLBE q-query security [GKW18]. Also, see Remark 4.1.

Definition 5.1 (q-query Normal Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query normal hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}_b) = b \ : \ \begin{array}{c}(1^\kappa, 1^n) \leftarrow \mathcal{A}(1^\lambda)\\ (\mathsf{pk},\mathsf{msk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n)\\ m \leftarrow \mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk})\\ b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\\ \mathsf{ct}_1 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (1,\perp,0))\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m,(j,\ell,\gamma))$ the index j must be equal to 1.

- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position j. That is, let $(j_1,\mathsf{id}_1),\ldots,(j_k,\mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$.

Definition 5.2 (q-query Index Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query index hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}) = b \ : \ \begin{array}{c}(1^\kappa, 1^n, i) \leftarrow \mathcal{A}(1^\lambda)\\ (\mathsf{pk},\mathsf{msk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n)\\ m \leftarrow \mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk})\\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i+b,\perp,0))\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m,(j,\ell,\gamma))$ the index j must be equal to either i or $i+1$.

- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position $j \in [n]$, and no key query of the form (i, id). That is, let $(j_1, \mathsf{id}_1), \ldots, (j_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$. And, $j_a \neq i$ for any a.

Definition 5.3 (q-query Upper Identity Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query upper identity hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}_b) = b \;:\; \begin{array}{c} (1^\kappa, 1^n, i, \ell, \beta) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ m \leftarrow \mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i+1, \perp, 0)) \\ \mathsf{ct}_1 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i, \ell, \beta)) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m, (j, \ell, \gamma))$ the index j must be equal to either i or $i+1$.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position $j \in [n]$, and no key query of the form (i, id) such that $\mathsf{id}_\ell = 1 - \beta$. That is, let $(j_1, \mathsf{id}_1), \ldots, (j_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$. And, for every a, $(\mathsf{id}_a)_\ell \neq 1 - \beta$ or $j_a \neq i$.

Definition 5.4 (q-query Lower Identity Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query lower identity hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}_b) = b \;:\; \begin{array}{c} (1^\kappa, 1^n, i, \ell, \beta) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ m \leftarrow \mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i, \perp, 0)) \\ \mathsf{ct}_1 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i, \ell, \beta)) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m, (j, \ell, \gamma))$ the index j must be equal to i.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position $j \in [n]$, and no key query of the form (i, id) such that $\mathsf{id}_\ell = \beta$. That is, let $(j_1, \mathsf{id}_1), \ldots, (j_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$. And, for every a, $(\mathsf{id}_a)_\ell \neq \beta$ or $j_a \neq i$.

Definition 5.5 (q-query Message Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query message hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}) = b \;:\; \begin{array}{c} (1^\kappa, 1^n) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{SplEnc}(\mathsf{key},\cdot,\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{SplEnc}(\mathsf{key}, m_b, (n+1, \perp, 0)) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m, (i, \ell, \gamma))$ the index i must be equal to $n+1$.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position i. That is, let $(i_1, \mathsf{id}_1), \ldots, (i_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then i_a and i_b must be distinct for all $a \neq b$.

5.2 Building Indexed EITT from EIPLBE

5.2.1 Construction

Consider an EIPLBE scheme $\mathsf{EIPLBE} = (\mathsf{EIPLBE.Setup}, \mathsf{EIPLBE.KeyGen}, \mathsf{EIPLBE.Enc}, \mathsf{EIPLBE.SplEnc}, \mathsf{EIPLBE.Dec})$ for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$. Below we provide our embedded identity TT construction with identical message and identity spaces. (Here we provide a transformation for TT schemes with secret key tracing, but the construction can be easily extended to work in the public tracing setting if the special encryption algorithm in the underlying EIPLBE scheme is public key as well.)

$\mathsf{Setup}(1^\lambda, 1^\kappa, n) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the EIPLBE setup as $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{EIPLBE.Setup}(1^\lambda, 1^\kappa, n)$, and outputs master secret-public-tracing key tuple $(\mathsf{msk}, \mathsf{pk}, \mathsf{key})$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, i) \to \mathsf{sk}_{i,\mathsf{id}}$. The key generation algorithm runs the EIPLBE key generation algorithm as $\mathsf{sk}_{i,\mathsf{id}} \leftarrow \mathsf{EIPLBE.KeyGen}(\mathsf{msk}, \mathsf{id}, i)$, and outputs secret key $\mathsf{sk}_{i,\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The encryption algorithm runs the EIPLBE encryption algorithm as $\mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m)$, and outputs ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. The decryption algorithm runs the EIPLBE decryption algorithm as $z \leftarrow \mathsf{EIPLBE.Dec}(\mathsf{sk}, \mathsf{ct})$, and outputs z.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1) \to T$. Let $\epsilon = 1/y$. First, consider the $\mathsf{Index\text{-}Trace}$ algorithm defined in Fig. 2. The sub-tracing algorithm simply tests whether the decoder box uses the user key for index i where i is one of the inputs provided to $\mathsf{Index\text{-}Trace}$. Now the tracing algorithm simply runs the $\mathsf{Index\text{-}Trace}$ algorithm for all indices $i \in [n]$, and for each index i where the $\mathsf{Index\text{-}Trace}$ algorithm outputs 1, the tracing algorithm adds index i to the *index-set* of traitors T^{index}.[10] Next, consider the $\mathsf{ID\text{-}Trace}$ algorithm defined in Fig. 3. The identity-tracing algorithm takes as input the index-set T^{index} and uses the decoder box to find the identity of the particular indexed user. Next, the tracing algorithm simply runs the $\mathsf{ID\text{-}Trace}$ algorithm for all indices $i \in T^{\text{index}}$, and for each index i where the $\mathsf{ID\text{-}Trace}$ algorithm does not output $\perp$, the tracing algorithm adds the output of the $\mathsf{ID\text{-}Trace}$ algorithm to the *identity-set* of traitors T.

Concretely, the algorithm runs as follows:

- Set $T^{\text{index}} := \emptyset$. For $i = 1$ to n:
  - Compute $(b, p, q) \leftarrow \mathsf{Index\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, i)$.
  - If $b = 1$, set $T^{\text{index}} := T^{\text{index}} \cup \{(i, p, q)\}$.
- Set $T := \emptyset$. For $(i, p, q) \in T^{\text{index}}$:
  - Compute $\mathsf{id} \leftarrow \mathsf{ID\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, (i, p, q))$.
  - Set $T := T \cup \{\mathsf{id}\}$.
- Output T.

Finally, it outputs the set T as the set of traitors.

Correctness. This follows directly from correctness of the underlying EIPLBE scheme.

[10]Technically, the set T^{index} constains tuples of the form (i, p, q) where i is an index and p, q are probabilities which are the estimations of successful decryption probability at index i and $i + 1$ (respectively).

```
Algorithm $\mathsf{Index\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, i)$

Inputs: Key $\mathsf{key}$, parameter $y$, messages $m_0, m_1$, index $i$
Output: $0/1$
Let $\epsilon = \lfloor 1/y \rfloor$. It sets $N = \lambda \cdot n/\epsilon$, and $\mathsf{count}_1 = \mathsf{count}_2 = 0$. For $j = 1$ to $N$, it computes the following:
  1. It chooses $b_j \leftarrow \{0,1\}$ and computes $\mathsf{ct}_{j,1} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_{b_j}, (i, \perp, 0))$ and sends $\mathsf{ct}_{j,1}$ to $D$.
     If $D$ outputs $b_j$, set $\mathsf{count}_1 = \mathsf{count}_1 + 1$, else set $\mathsf{count}_1 = \mathsf{count}_1 - 1$.
  2. It chooses $c_j \leftarrow \{0,1\}$ and computes $\mathsf{ct}_{j,2} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_{c_j}, (i+1, \perp, 0))$ and sends $\mathsf{ct}_{j,2}$ to
     $D$. If $D$ outputs $c_j$, set $\mathsf{count}_2 = \mathsf{count}_2 + 1$, else set $\mathsf{count}_2 = \mathsf{count}_2 - 1$.
If $\frac{\mathsf{count}_1 - \mathsf{count}_2}{N} > \frac{\epsilon}{4n}$, output $(1, \frac{\mathsf{count}_1}{N}, \frac{\mathsf{count}_2}{N})$, else output $(0, \perp, \perp)$.
```

Figure 2: Index-Trace

```
Algorithm $\mathsf{ID\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, (i, p, q))$

Inputs: Key $\mathsf{key}$, parameter $y$, messages $m_0, m_1$, index $i$, probabilities $p, q$
Output: $\mathsf{id} \in \{0,1\}^\kappa$
Let $\epsilon = \lfloor 1/y \rfloor$. It sets $N = \lambda \cdot n/\epsilon$, and $\mathsf{count}_\ell = 0$ for $\ell \in [\kappa]$. For $\ell = 1$ to $\kappa$, it proceeds as follows:
  1. For $j = 1$ to $N$, it computes the following:
     (a) It chooses $b_j \leftarrow \{0,1\}$ and computes $\mathsf{ct}_j \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_{b_j}, (i, \ell, 0))$ and sends $\mathsf{ct}_j$ to $D$.
         If $D$ outputs $b_j$, set $\mathsf{count}_\ell = \mathsf{count}_\ell + 1$, else set $\mathsf{count}_\ell = \mathsf{count}_\ell - 1$.
Next, let $\mathsf{id}$ be an empty string. For $\ell = 1$ to $\kappa$, do the following:
  1. If $\frac{p+q}{2} > \frac{\mathsf{count}_\ell}{N}$, set $\mathsf{id}_\ell = 0$. Else set $\mathsf{id}_\ell = 1$.
Finally, output $\mathsf{id}$.
```

Figure 3: Index-Trace

Efficiency. If the scheme EIPLBE = (EIPLBE.Setup, EIPLBE.KeyGen, EIPLBE.Enc, EIPLBE.SplEnc, EIPLBE.Dec) is a EIPLBE scheme with (T-s, T-e, T-$\widetilde{\text{e}}$, T-k, T-d, S-c, S-k)-efficiency, then the scheme TT = (Setup, KeyGen, Enc, Dec, Trace) is a (indexed keygen, public/private)-embedded identity tracing scheme with (T-s′, T-e′, T-k′, T-d′, T-t′, S-c′, S-k′)-efficiency, where the efficiency measures are related as follows:

- $\mathsf{T\text{-}s}'(\lambda, \kappa, n) = \mathsf{T\text{-}s}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}k}'(\lambda, \kappa, n) = \mathsf{T\text{-}k}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}e}'(\lambda, \kappa, n) = \mathsf{T\text{-}e}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}d}'(\lambda, \kappa, n) = \mathsf{T\text{-}d}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}t}'(\lambda, \kappa, n, y) = (2n + \kappa) \cdot \lambda \cdot y \cdot n$,
- $\mathsf{S\text{-}c}'(\lambda, \kappa, n) = \mathsf{S\text{-}c}(\lambda, \kappa, n)$,
- $\mathsf{S\text{-}k}'(\lambda, \kappa, n) = \mathsf{S\text{-}k}(\lambda, \kappa, n)$.

5.2.2 Security

In this section, we prove security of our construction. Formally, we prove the following.

Theorem 5.1. If the scheme EIPLBE = (EIPLBE.Setup, EIPLBE.KeyGen, EIPLBE.Enc, EIPLBE.SplEnc, EIPLBE.Dec) is a 1-query secure EIPLBE scheme as per Definitions 5.1 to 5.5, then the scheme $\mathcal{T}$ = (Setup, KeyGen, Enc, Dec, Trace) is a secure (indexed keygen, public/private)-embedded identity tracing scheme as per Definitions 4.2 and B.2.

We prove the above theorem in two parts. First, we prove IND-CPA security of the above construction. Later we argue correctness of tracing to complete the proof of security. Note that in the proof we use the fact that n is bounded by a polynomial in the security parameter. (See Remark 4.1.)

IND-CPA Security. We would like to point out that the scheme $\mathcal{T}$ is IND-CPA secure even if the EIPLBE scheme satisfies only 0-query security. In other words, we do not need the scheme to achieve 1-query security for arguing IND-CPA security. At a high level, the proof of IND-CPA security is identical to that used for proving IND-CPA security of (standard) traitor tracing systems from (standard) private linear broadcast encryption scheme [BSW06]. Below we provide a high level sketch.

Lemma 5.1. If the scheme EIPLBE is a 0-query secure EIPLBE scheme as per Definitions 5.1, 5.2 and 5.5, then the scheme $\mathcal{T}$ is an IND-CPA secure (indexed keygen, public/private)-embedded identity tracing scheme as per Definition 4.2.

Proof. We will construct a sequence of $2n + 4$ hybrid experiments to prove IND-CPA security. The first experiment, that is Hybrid H_0, is exactly the IND-CPA game.

Hybrid H_0 : In this experiment, the challenger sends public key pk, receives m_0, m_1 from $\mathcal{A}$ and sends $\mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m_0)$ to $\mathcal{A}$.

Hybrid $H_{i,b}$ **(for** $i \in [n+1], b \in \{0,1\}$**)** : This experiment is identical to the IND-CPA experiment, except that the adversary, after sending challenge messages m_0, m_1, receives $\mathsf{ct} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_b, (i, \perp, 0))$.

Hybrid H_1 : In this experiment, the challenger sends public key pk, receives m_0, m_1 from $\mathcal{A}$ and sends $\mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m_1)$ to $\mathcal{A}$.

For any PPT adversary $\mathcal{A}$, let $p_{\mathcal{A},x}(\cdot)$ be a function of λ that denotes the probability of $\mathcal{A}$ outputting 0 in Hybrid H_x. Note that $p_{\mathcal{A},0} - p_{\mathcal{A},1}$ is the advantage of $\mathcal{A}$ in the IND-CPA security game.

Claim 5.1. If the scheme EIPLBE is a 0-query normal hiding secure EIPLBE scheme as per Definition 5.1, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $b \in \{0,1\}$, $|p_{\mathcal{A},b} - p_{\mathcal{A},1,b}| \leq \mathsf{negl}(\lambda)$.

This follows from 0-query normal hiding security of EIPLBE.

Claim 5.2. If the scheme EIPLBE is a 0-query index hiding secure EIPLBE scheme as per Definition 5.2, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $(i, b) \in [n] \times \{0,1\}$, $|p_{\mathcal{A},i,b} - p_{\mathcal{A},i+1,b}| \leq \mathsf{negl}(\lambda)$.

This follows from 0-query index hiding security of EIPLBE.

Claim 5.3. If the scheme EIPLBE is a 0-query message hiding secure EIPLBE scheme as per Definition 5.5, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|p_{\mathcal{A},n+1,0} - p_{\mathcal{A},n+1,1}| \leq \mathsf{negl}(\lambda)$.

This follows from 0-query message hiding security of EIPLBE.

Combining the above claims, we get proof of Lemma 5.1.

□

Correctness of Tracing. Next, we show that the false trace probability is bounded by a negligible function, and the correct trace probability is close to the probability of $\mathcal{A}$ outputting an ϵ-successful decoding box for some non-negligible ϵ.

First, we introduce some notations. Fix some public-master secret key pair $(\mathsf{pk}, \mathsf{msk})$. Given any pirate decoder box D and messages m_0, m_1, for any $i \in [n+1]$, $\ell \in [\kappa]$, let

$$p^D_{i,\perp} = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_b, (i, \perp, 0))]$$

$$p^D_{i,\ell} = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_b, (i, \ell, 0))]$$

$$p^D_{\text{nrml}} = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m_b)]$$

where the probability is taken over random coins of decoder D as well as the randomness used during encryption.

False Trace Probability. First, we show that the tracing algorithm never falsely accuses any user with non-negligible probability. Formally, we prove the following.

Theorem 5.2. If the scheme EIPLBE is a 1-query secure EIPLBE scheme as per Definitions 5.1 to 5.5, then for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$,

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathsf{negl}(\lambda),$$

where $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ is as defined in Definition B.2.

Proof. Let $S \subseteq [n] \times \{0,1\}^\kappa$ be the set of index-identity pairs queried by the adversary $\mathcal{A}$ for secret keys, $S_{\text{index}} \subseteq [n]$ be the set of indices queried by the adversary $\mathcal{A}$ for secret keys, and let D be the decoder box output by $\mathcal{A}$.

In the sequel we skip the dependence of $\epsilon(\cdot)$ on λ for simplicity of notation. For $i \in [n], \ell \in [\kappa]$, we define events

$$\begin{aligned}
\mathsf{Diff\text{-}Adv}_i^D &: p_{i,\perp}^D - p_{i+1,\perp}^D > \epsilon/8n \\
\mathsf{Diff\text{-}Adv}_{i,\ell,\text{lwr}}^D &: p_{i,\perp}^D - p_{i,\ell}^D > \epsilon/16n \\
\mathsf{Diff\text{-}Adv}_{i,\ell,\text{upr}}^D &: p_{i,\ell}^D - p_{i+1,\perp}^D > \epsilon/16n \\
\mathsf{Diff\text{-}Adv}^D &: \begin{array}{c} \bigvee\limits_{i \in [n] \setminus S_{\text{index}}} \mathsf{Diff\text{-}Adv}_i^D \\ \bigvee\limits_{\substack{(i,\mathsf{id}) \in S,\ \ell \in [\kappa] \\ \text{s.t. } \mathsf{id}_\ell = 1}} \mathsf{Diff\text{-}Adv}_{i,\ell,\text{lwr}}^D \quad \bigvee\limits_{\substack{(i,\mathsf{id}) \in S,\ \ell \in [\kappa] \\ \text{s.t. } \mathsf{id}_\ell = 0}} \mathsf{Diff\text{-}Adv}_{i,\ell,\text{upr}}^D \end{array}
\end{aligned}$$

For simplicity of notation, we will drop dependence on decoder D whenever clear from context. Next, note that the probability of the event *false trace* can be rewritten (using union bound) as follows by conditioning on the events defined above

$$\begin{aligned}
\Pr[\mathsf{Fal\text{-}Tr}] \leq\ & \Pr\left[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}\right] + \sum_{i \in [n]} \Pr\left[i \notin S_{\text{index}} \wedge \mathsf{Diff\text{-}Adv}_i\right] \\
& + \sum_{(i,\ell) \in [n] \times [\kappa]} \Pr\left[\exists \mathsf{id} \in \{0,1\}^\kappa \text{ s.t. } (i,\mathsf{id}) \in S \wedge \left(\begin{array}{c} (\mathsf{Diff\text{-}Adv}_{i,\ell,\text{lwr}} \wedge \mathsf{id}_\ell = 1) \vee \\ (\mathsf{Diff\text{-}Adv}_{i,\ell,\text{upr}} \wedge \mathsf{id}_\ell = 0) \end{array} \right)\right].
\end{aligned}$$

We will show that each of these terms is bounded by a negligible function. We start by bounding the first term.

Lemma 5.2. For every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}] \leq \mathsf{negl}_1(\lambda).$$

Proof. The proof of this lemma follows from Chernoff bounds, and is similar to those provided in [GKW18, Lemma 4.4] and [GKRW18, Lemma 5.3]. Here we sketch the high level idea.

Note that event $\mathsf{Fal\text{-}Tr}$ occurs iff the tracing algorithm outputs a user identity which was not key queried by the adversary. Recall that the tracing algorithm takes a two-step approach. It proceeds by first tracing the key indices of the corrupted keys, and then it traces the corresponding identity. Now there are two sources of error in incorrect tracing. First, during step one of tracing the algorithm might incorrectly include some index $i \notin S_{\text{index}}$ in the index-set of traitors T^{index}. Second, during step two it may output a non-corrupt identity id for some index $i \in S_{\text{index}}$, that is for some $i \in S_{\text{index}}$ the $\mathsf{ID\text{-}Trace}$ algorithm traces the id

incorrectly at at least one bit position. Thus, we could write the following (using union bound), where sets T and T^{index} are as defined in the description of the tracing algorithm,

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}] \leq \sum_{i \in [n]} \Pr\left[\mathsf{Fal\text{-}Tr} \wedge i \notin S_{\text{index}} \wedge (\exists\, p, q : (i, p, q) \in T^{\text{index}}) \mid \overline{\mathsf{Diff\text{-}Adv}}\right]$$
$$+ \sum_{(i,\ell) \in [n] \times [\kappa]} \Pr\left[\mathsf{Fal\text{-}Tr} \wedge \exists \mathsf{id}, \widetilde{\mathsf{id}} : (i, \mathsf{id}) \in S \wedge \widetilde{\mathsf{id}} \in T \wedge \mathsf{id}_\ell \neq \widetilde{\mathsf{id}}_\ell \mid \overline{\mathsf{Diff\text{-}Adv}}\right].$$

In the above expression, the first term on the right side bounds the type 1 error (i.e., faulty step one tracing) and the second term bounds the type 2 error (i.e., faulty step two tracing).

Now let us analyze the first term. Note that if event $\overline{\mathsf{Diff\text{-}Adv}}$ occurs then it implies that for every $i \notin S_{\text{index}}$ event $\overline{\mathsf{Diff\text{-}Adv}_i}$ occurred. Thus, it must hold that for every $i \in [n]$

$$\Pr\left[i \notin S_{\text{index}} \wedge (\exists\, p, q : (i, p, q) \in T^{\text{index}}) \mid \overline{\mathsf{Diff\text{-}Adv}}\right] \leq 2^{-O(\lambda)}.$$

This follows from a Chernoff bound since $\overline{\mathsf{Diff\text{-}Adv}_i}$ states that $p_{i,\perp} - p_{i+1,\perp} \leq \epsilon/8n$ and event $(\exists\, p, q : (i, p, q) \in T^{\text{index}})$ suggests that $\hat{p}_{i,\perp} - \hat{p}_{i+1,\perp} > \epsilon/4n$ where $\hat{p}$ denotes the corresponding estimate computed by the tracing algorithm.

Next, let us analyze the second term. For a fixed (i, ℓ), the probability term corresponds to the event that the $\mathsf{ID\text{-}Trace}$ algorithm outputs the traitor identity $\widetilde{\mathsf{id}}$ such that $\mathsf{id}_\ell \neq \widetilde{\mathsf{id}}_\ell$ where the adversary makes a key query for index-identity pair (i, id). (Recall that the adversary is allowed to receive at most one key per index. See Definition B.2.) Concretely, by conditioning on the event $\overline{\mathsf{Diff\text{-}Adv}}$ we get that for every $(i, \mathsf{id}) \in S, \ell \in [\kappa]$, event $\overline{\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{X}}}$ always occurs where $\mathrm{X} = \mathrm{lwr}$ if $\mathsf{id}_\ell = 1$ else $\mathrm{X} = \mathrm{upr}$. Thus, it must hold that for every $(i, \mathsf{id}) \in S, \ell \in [\kappa]$

$$\Pr\left[\exists \mathsf{id}, \widetilde{\mathsf{id}} : (i, \mathsf{id}) \in S \wedge \widetilde{\mathsf{id}} \in T \wedge \mathsf{id}_\ell \neq \widetilde{\mathsf{id}}_\ell \mid \overline{\mathsf{Diff\text{-}Adv}}\right] \leq 2^{-O(\lambda)}.$$

For simplicity, fix (i, id, ℓ) and let $\mathsf{id}_\ell = 1$. Then the above statement follows from a Chernoff bound since we know that event $\overline{\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}}}$ occurs, thus we have that $p_{i,\perp} - p_{i,\ell} \leq \epsilon/16n$ and event $\widetilde{\mathsf{id}} \in T \wedge \widetilde{\mathsf{id}}_\ell = 0$ suggests that $\hat{p}_{i,\perp} - \hat{p}_{i,\ell} > \epsilon/8n$ where $\hat{p}$ denotes the corresponding estimate computed by the tracing algorithm.

Therefore, combining all the above claims we get that

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}] \leq n \cdot 2^{-O(\lambda)} + n \cdot \kappa \cdot 2^{-O(\lambda)} = \mathsf{negl}_1(\lambda).$$

□

Lemma 5.3. If the scheme EIPLBE is a 1-query index hiding secure EIPLBE scheme as per Definition 5.2, then for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$ and $i \in [n]$,

$$\Pr[i \notin S_{\text{index}} \wedge \mathsf{Diff\text{-}Adv}_i] \leq \mathsf{negl}_2(\lambda),$$

where n is the index bound chosen, and S_{index} is the set of indices queried by $\mathcal{A}$.

Proof. The proof of this lemma is similar to those provided in [GKW18, Lemma 4.5] and [GKRW18, Lemma 5.4]. □

Lemma 5.4. If the scheme EIPLBE is a 1-query lower and upper identity hiding secure EIPLBE scheme as per Definitions 5.3 and 5.4, then for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\mathsf{negl}_3(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$ and $i \in [n], \ell \in [\kappa]$,

$$\Pr\left[\exists \mathsf{id} \in \{0,1\}^\kappa \text{ s.t. } (i, \mathsf{id}) \in S \wedge \begin{pmatrix} (\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}} \wedge \mathsf{id}_\ell = 1) \vee \\ (\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{upr}} \wedge \mathsf{id}_\ell = 0) \end{pmatrix}\right] \leq \mathsf{negl}_3(\lambda),$$

where n is the index bound chosen, and S is the set of index-identity pairs queried by $\mathcal{A}$.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible functions $\epsilon(\cdot), \delta(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, there exists an $i^* \in [n], \ell^* \in [\kappa]$ s.t.

$$\Pr\left[\exists \mathsf{id} \in \{0,1\}^\kappa \text{ s.t. } (i^*, \mathsf{id}) \in S \wedge \left(\begin{array}{c} (\mathsf{Diff\text{-}Adv}_{i^*,\ell^*,\mathrm{lwr}} \wedge \mathsf{id}_{\ell^*} = 1) \vee \\ (\mathsf{Diff\text{-}Adv}_{i^*,\ell^*,\mathrm{upr}} \wedge \mathsf{id}_{\ell^*} = 0) \end{array} \right)\right] \geq \delta(\lambda).$$

Then we can use $\mathcal{A}$ to build a PPT reduction algorithm $\mathcal{B}$ that breaks the upper/lower identity hiding security property of EIPLBE. The reduction algorithm $\mathcal{B}$ first receives $1^n, 1^\kappa$ from the adversary. It chooses a random index $i \leftarrow [n]$, position $\ell \in [\kappa]$, and value $b \in \{0,1\}$, and sends the challenge index-position-value tuple $(i, \ell, 0)$ and $(1^n, 1^\kappa)$ to the EIPLBE challenger.[11] (In other words, the reduction algorithm randomly guesses the index-position pair (i^*, ℓ^*) as well as if $b = 0$ it interacts with EIPLBE lower identity hiding challenger, otherwise if $b = 1$ it interacts with EIPLBE upper identity hiding challenger.) It then receives the EIPLBE public key pk from the challenger, which it sends to $\mathcal{A}$. The adversary $\mathcal{A}$ then queries for secret keys. If $\mathcal{A}$ key queries for index-identity pair (j, id) where $j = i$ and $\mathsf{id}_\ell = b$, then $\mathcal{B}$ aborts and sends a random guess to the EIPLBE challenger. Else, on key query for (j, id) from $\mathcal{A}$, reduction algorithm $\mathcal{B}$ forwards (j, id) to the EIPLBE challenger and forwards the challenger's response to the adversary. After all key queries, the adversary outputs a decoding box D and messages m_0, m_1 to $\mathcal{B}$. $\mathcal{B}$ then chooses two bits α, β uniformly at random, i.e. $\alpha, \beta \leftarrow \{0,1\}$. Next, $\mathcal{B}$ sends message m_α as its challenge message, and receives challenge ciphertext ct^* from EIPLBE challenger. It also queries the EIPLBE challenger for a special-encryption of m_α for index-position-value tuple $(i, \ell, 0)$ if $\beta = 0$, else for $(i + b, \perp, 0)$. Let ct be the challenger's response. Finally, $\mathcal{B}$ runs decoder box D on ct and ct^* independently, and if $D(\mathsf{ct}) = D(\mathsf{ct}^*)$, it outputs $b' = \beta$, else it outputs $b' = 1 - \beta$ as its guess.

First, note that $\mathcal{B}$ is an admissible adversary in the upper/lower identity hiding security game (if $b = 0$ then 'lower', else 'upper' respectively). This is because $\mathcal{B}$ does not query the challenger for secret key on index-identity pair (j, id) such that $j = i$ and $\mathsf{id}_\ell = b$. Additionally, it only makes a single special-encryption query on index-position-value tuple $(i, \ell, 0)$ or $(i+b, \perp, 0)$. Finally, by an analysis similar to that in [GKW18, Lemma 4.1, 4.5] and [GKRW18, Lemma 5.4], it follows that the advantage of the reduction algorithm is at least $\frac{\delta}{2\kappa n}\left(\frac{\epsilon}{16n}\right)^2$. Thus, the lemma follows. □

From the above lemmas, it follows that the probability of false trace is at most $\mathsf{negl}_1(\lambda) + n \cdot \mathsf{negl}_2(\lambda) + n \cdot \kappa \cdot \mathsf{negl}_3(\lambda)$, thus theorem follows.

□

Correct Trace Probability. Now we show that whenever the adversary outputs a good decoder, then with all but negligible probability the tracing algorithm outputs a non-empty set T. Combining this with Theorem 5.2, we get that the tracing algorithm correctly traces. Formally, we show the following.

Theorem 5.3. If the scheme EIPLBE is a 1-query secure EIPLBE scheme as per Definitions 5.1 to 5.5, then for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$,

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}(\lambda)$$

where $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ and $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\cdot)$ are as defined in Definition B.2.

Proof. Let us start by analyzing the probability that tracing algorithm outputs a non-empty set T. First, we know that if event $\mathsf{Good\text{-}Decoder}$ occurs, then $p^D_{\mathrm{nrml}} \geq 1/2 + \epsilon$ for some non-negligible ϵ. Next, let $S_{\mathrm{index}} \subseteq [n]$ be the set of indices $i \in [n]$ such that $p^D_{i,\perp} - p^D_{i+1,\perp} > \epsilon/2n$. By using Chernoff bounds similar to that in Lemma 5.2, we get that

$$\forall\ i \in S_{\mathrm{index}}, \quad \Pr\left[\hat{p}^D_{i,\perp} - \hat{p}^D_{i+1,\perp} < \epsilon/4n\right] \leq 2^{-O(\lambda)} = \mathsf{negl}_1(\lambda), \tag{1}$$

[11]Note that both n and κ are outputted in unary, thus they are some polynomials in the security parameter.

where $\hat{p}$ denotes the corresponding estimate computed by the tracing algorithm.

Note that by 1-query message hiding security of the underlying EIPLBE scheme, we have that $p^D_{n+1,\perp} \leq 1/2 + \mathsf{negl}_2(\lambda)$ for some negligible function $\mathsf{negl}_2(\cdot)$. Also, by 1-query normal hiding security, we have that $p^D_{\text{nrml}} - p^D_{1,\perp} \leq \mathsf{negl}_3(\lambda)$ for some negligible function $\mathsf{negl}_3(\cdot)$. Thus, we can write that

$$p^D_{1,\perp} - p^D_{n+1,\perp} \geq \epsilon - \mathsf{negl}_2(\lambda) - \mathsf{negl}_3(\lambda) > \epsilon/2.$$

Given this we can conclude that the set S_{index} (as defined above) must be non-empty whenever event Good-Decoder occurs. Combining this with Eq. (1), we get that if event Good-Decoder occurs then with all-but-negligible probability

$$T^{\text{index}} \neq \emptyset, \qquad \text{and} \qquad \forall\ (i,p,q) \in T^{\text{index}}:\ p - q > \frac{\epsilon}{4n}$$

where T^{index} is as defined in the tracing algorithm.

Looking back at Fig. 3, we observe that for every tuple (i,p,q) the ID-Trace algorithm always outputs some identity id. This is because the algorithm simply checks for every $\ell \in [\kappa]$, either $p^D_{i,\ell} > (p+q)/2$ and the algorithm sets $\mathsf{id}_\ell = 1$, otherwise it sets $\mathsf{id}_\ell = 0$. Thus, this implies the following:

$$T^{\text{index}} \neq \emptyset \implies T \neq \emptyset.$$

Therefore, we can write the following

$$\Pr[T \neq \emptyset] \geq (1 - n \cdot \mathsf{negl}_1(\lambda)) \cdot \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}(\lambda).$$

Finally, combining with Theorem 5.2, we get that

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}(\lambda).$$

This concludes the proof. □

6 Building EIPLBE from Public Key Encryption

Let $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a public key encryption scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$. Below we provide an EIPLBE scheme for same message space and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$.

$\mathsf{Setup}(1^\lambda, 1^\kappa, n) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the PKE setup $2n \cdot \kappa$ times as

$$(\mathsf{sk}_{i,\ell,b}, \mathsf{pk}_{i,\ell,b}) \leftarrow \mathsf{PKE.Setup}(1^\lambda), \qquad \text{for } (i,\ell,b) \in [n] \times [\kappa] \times \{0,1\}$$

and outputs sets $\mathsf{pk} = \{\mathsf{pk}_{i,\ell,b}\}_{(i,\ell,b)\in[n]\times[\kappa]\times\{0,1\}}$, $\mathsf{msk} = \{\mathsf{sk}_{i,\ell,b}\}_{(i,\ell,b)\in[n]\times[\kappa]\times\{0,1\}}$, and $\mathsf{key} = \mathsf{pk}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa, i \in [n]) \to \mathsf{sk}$. Let $\mathsf{msk} = \{\mathsf{sk}_{j,\ell,b}\}_{j,\ell,b}$. The key generation algorithm outputs the secret key as $\mathsf{sk} = \left(i, \mathsf{id}, \{\mathsf{sk}_{i,\ell,\mathsf{id}_\ell}\}_{\ell\in[\kappa]}\right)$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. Let $\mathsf{pk} = \{\mathsf{pk}_{i,\ell,b}\}_{i,\ell,b}$. The encryption algorithm first chooses $n \cdot (\kappa - 1)$ random messages as $r_{i,\ell} \leftarrow \mathcal{M}$ for $(i,\ell) \in [n] \times [\kappa - 1]$. Next, for every i, it sets $r_{i,\kappa} = m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i,\ell}\right)$. It then encrypts messages $r_{i,\ell}$ under key $\mathsf{pk}_{i,\ell,b}$ as follows:

$$\mathsf{ct}_{i,\ell,b} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{i,\ell,b}, r_{i,\ell}), \qquad \text{for } (i,\ell,b) \in [n] \times [\kappa] \times \{0,1\}$$

Finally, it outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_{i,\ell,b})_{i,\ell,b}$.

$\mathsf{SplEnc}(\mathsf{key}, m, (i^*, \ell^*, b^*)) \to \mathsf{ct}$. Let $\mathsf{key} = \left\{\mathsf{pk}_{i,\ell,b}\right\}_{i,\ell,b}$. The encryption algorithm first chooses $n \cdot (\kappa - 1)$ random messages as $r_{i,\ell} \leftarrow \mathcal{M}$ for $(i, \ell) \in [n] \times [\kappa - 1]$. Next, it sets $r_{i,\kappa}$ as:

$$r_{i,\kappa} = \begin{cases} m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i,\ell}\right) & i \geq i^* \\ \leftarrow \mathcal{M} & \text{otherwise} \end{cases}$$

where '$\leftarrow \mathcal{M}$' denotes sampling $r_{i,\ell}$ as a random message. Now it sets messages $\widetilde{r}_{i,\ell,b}$ as:

$$\widetilde{r}_{i,\ell,b} = \begin{cases} r_{i,\ell} & \text{if } (i, \ell, b) \neq (i^*, \ell^*, b^*) \\ \leftarrow \mathcal{M} & \text{otherwise} \end{cases}$$

It then encrypts messages $\widetilde{r}_{i,\ell,b}$ under key $\mathsf{pk}_{i,\ell,b}$ as follows:

$$\mathsf{ct}_{i,\ell,b} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{i,\ell,b}, \widetilde{r}_{i,\ell,b}), \qquad \text{for } (i, \ell, b) \in [n] \times [\kappa] \times \{0, 1\}$$

Finally, it outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_{i,\ell,b})_{i,\ell,b}$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. Let $\mathsf{sk} = (i, \mathsf{id}, \{\mathsf{sk}_\ell\}_\ell)$ and $\mathsf{ct} = (\mathsf{ct}_{j,\ell,b})_{j,\ell,b}$. The decryption algorithm runs the PKE decryption on ciphertexts $\{\mathsf{ct}_{i,\ell,\mathsf{id}_\ell}\}_\ell$ as follows:

$$z_\ell \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_\ell, \mathsf{ct}_{i,\ell,\mathsf{id}_\ell}), \qquad \text{for } \ell \in [\kappa].$$

Finally, it outputs $\bigoplus_{\ell=1}^{\kappa} z_\ell$.

Correctness. This follows directly from correctness of the underlying PKE scheme. Below we briefly sketch the main points.

First, note that the EIPLBE (normal) encryption algorithm computes each ciphertext $\mathsf{ct}_{i,\ell,b}$ such that it encrypts message $r_{i,\ell}$ under public key $\mathsf{pk}_{i,\ell,b}$ with the condition that, for every i, $\bigoplus_{\ell=1}^{\kappa} r_{i,\ell} = m$ where m is the message encrypted. Now the secret key for user j with identity id consists of PKE secret keys $\mathsf{sk}_{j,\ell,\mathsf{id}_\ell}$. Thus, the decryption algorithm correctly reconstructs the message if the underlying PKE scheme is correct.

Second, the EIPLBE special-encryption algorithm, on input (i^*, ℓ^*, b^*), computes each ciphertext $\mathsf{ct}_{i,\ell,b}$ such that it encrypts message $\widetilde{r}_{i,\ell,b}$ under public key $\mathsf{pk}_{i,\ell,b}$ where we know that for

$$((i \geq i^* + 1) \ \vee \ (i^*, \ell^*) = (i, \perp) \ \vee \ (i^*, \mathsf{id}_{\ell^*}) = (i, 1 - b^*)) \implies \widetilde{r}_{i,\ell,b} = r_{i,\ell}.$$

And since we know that for every $i \geq i^*$, $\bigoplus_{\ell=1}^{\kappa} r_{i,\ell} = m$ where m is the message encrypted. Thus, correctness of special-encryption follows from the correctness of the underlying PKE scheme.

Efficiency. Let the public key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}d}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)-efficient, then the EIPLBE scheme $\mathsf{EIPLBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec})$ is ($\mathsf{T\text{-}s}'$, $\mathsf{T\text{-}e}'$, $\mathsf{T\text{-}\widetilde{e}}'$, $\mathsf{T\text{-}k}'$, $\mathsf{T\text{-}d}'$, $\mathsf{S\text{-}c}'$, $\mathsf{S\text{-}k}'$)-efficient, where the efficiency measures are related as follows:

- $\mathsf{T\text{-}s}'(\lambda, \kappa, n) = 2n \cdot \kappa \cdot \mathsf{T\text{-}s}(\lambda)$,
- $\mathsf{T\text{-}k}'(\lambda, \kappa, n) = O(2n \cdot \kappa \cdot \lambda)$,
- $\mathsf{T\text{-}e}'(\lambda, \kappa, n) = 2n \cdot \kappa \cdot \mathsf{T\text{-}e}(\lambda)$,
- $\mathsf{T\text{-}\widetilde{e}}'(\lambda, \kappa, n) = 2n \cdot \kappa \cdot \mathsf{T\text{-}e}(\lambda)$,
- $\mathsf{T\text{-}d}'(\lambda, \kappa, n) = \kappa \cdot \mathsf{T\text{-}d}(\lambda)$,
- $\mathsf{S\text{-}c}'(\lambda, \kappa, n) = 2n \cdot \kappa \cdot \mathsf{S\text{-}c}(\lambda)$,
- $\mathsf{S\text{-}k}'(\lambda, \kappa, n) = \kappa \cdot \mathsf{S\text{-}k}(\lambda)$.

6.1 Security

In this section, we prove security of our construction. Formally, we prove the following.

Theorem 6.1. If the scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is an IND-CPA secure PKE scheme, then the scheme $\mathsf{EIPLBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec})$ described above is a secure EIPLBE as per Definitions 5.1 to 5.5.

We prove the above theorem in parts where we prove that our construction satisfies all five security properties. The proof of each security property follows directly from the IND-CPA security of the underlying PKE scheme. Before we sketch the main ideas behind the proof, we would like to highlight that since the EIPLBE scheme described above has a public key special-encryption algorithm thus the adversary does not need to query the EIPLBE challenger for special-encryption queries since the adversary can simulate them on its own. Therefore, throughout this section we consider that the adversary queries the challenger for secret keys *only.*

6.1.1 Normal Hiding

This follows directly from our construction since when the special-encryption algorithm is run on input (i^*, ℓ^*, b^*) with $i^* = 1$ and $\ell^* = \perp$, then $\widetilde{r}_{i,\ell,b} = r_{i,\ell}$ for all i, ℓ, b as the condition $(i > i^*) \vee (i = i^* \wedge (\ell \neq \ell^* \vee b \neq b^*))$ is equivalent to $i \geq 1$ which is always true. Thus, the distributions of (normal) ciphertexts and (special-encryption) ciphertexts is identical.

6.1.2 Index Hiding

Recall that index hiding property requires that special-encryption of message m for index-position-value tuple $(i^*, \perp, 0)$ is indistinguishable from special-encryption of message m for index-position-value tuple $(i^*+1, \perp, 0)$ if the adversary does not receice any key for index i^*. Now the proof of index hiding follows from a simple hybrid experiment. The main idea is that we will use IND-CPA security to switch the sub-ciphertexts $\mathsf{ct}_{i^*,\kappa,0}, \mathsf{ct}_{i^*,\kappa,1}$ to encryptions of a common random message instead of encryptions of $m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i^*,\ell}\right)$. This can be done since the adversary is not allowed to key query for index i^*, thus the adversary does not receive secret keys $\mathsf{sk}_{i^*,\kappa,0}, \mathsf{sk}_{i^*,\kappa,1}$ and therefore using IND-CPA security we can indistinguishably switch the corresponding ciphertexts to encrypt a random message $r_{i^*,\kappa} \leftarrow \mathcal{M}$.

We construct a sequence of 3 hybrid experiments. The first experiment, Hybrid H_0, is exactly the index hiding security game in which the challenge ciphertext is an encryption for index-position-value tuple $(i^*, \perp, 0)$. The next hybrid H_1 is identical to H_0 except the ciphertext $\mathsf{ct}_{i^*,\kappa,0}$ encrypts a random message $r_{i^*,\kappa}$ instead of $m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i^*,\ell}\right)$. And, in the final hybrid H_2, both ciphertext $\mathsf{ct}_{i^*,\kappa,0}$ and $\mathsf{ct}_{i^*,\kappa,1}$ encrypt a random message $r_{i^*,\kappa}$ instead of $m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i^*,\ell}\right)$. Thus, index hiding security follows from IND-CPA of the PKE scheme.

6.1.3 Upper/Lower Identity Hiding

The proof of upper/lower identity hiding again follows directly from the IND-CPA security of the underlying PKE construction and is similar to the previous proof. Let (i^*, ℓ^*, b^*) be the adversary's challenge index-position-value tuple. In the case of upper identity security, the challenger uses IND-CPA security to switch ciphertext $\mathsf{ct}_{i^*,\ell^*,1-b^*}$ from encryption of $m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i^*,\ell}\right)$ to encryption of $r_{i^*,\kappa}$ where $r_{i^*,\kappa}$ is the message encrypted in the ciphertext $\mathsf{ct}_{i^*,\ell^*,b^*}$. This follows from the fact that the adversary is not allowed to query (j, id) for secret key where $j = i^*$ and $\mathsf{id}_{\ell^*} = 1 - b^*$.

Similarly, for lower identity hiding security, the challenger uses IND-CPA to switch $\mathsf{ct}_{i^*,\ell^*,b^*}$ from encryption of a random message $r_{i^*,\kappa}$ to encryption of $m \oplus \left(\bigoplus_{\ell=1}^{\kappa-1} r_{i^*,\ell}\right)$. This again follows from the fact that the adversary is not allowed to query (j, id) for secret key where $j = i^*$ and $\mathsf{id}_{\ell^*} = b^*$.

6.1.4 Message Hiding

This follows directly from our construction since the special-encryption algorithm when run on input $(n+1, \perp, b^*)$ computes sub-ciphertexts $\mathsf{ct}_{i,\ell,b}$ as encryptions of completely random and independent messages (for all i, ℓ).[12] (Note that for a any i, ℓ, $\mathsf{ct}_{i,\ell,0}$ and $\mathsf{ct}_{i,\ell,1}$ encrypt the same message.) Thus, the ciphertext is completely independent of the message m, therefore the distributions of special-encryptions on any two messages m_0 and m_1 is identical.

7 Building EIPLBE using Bilinear Maps

Let Gen be a bilinear group generator of composite order. Below we provide our EIPLBE scheme based on bilinear maps. Below we are using a notation similar to that used in [BW06, Section 4.3-4.4] for indexing users.

$\mathsf{Setup}(1^\lambda, 1^\kappa, n) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. Let $\widetilde{m} = \left\lceil \sqrt{\frac{n}{\kappa}} \right\rceil$ and $m = \left\lceil \frac{n}{\widetilde{m}} \right\rceil$.[13] The setup algorithm samples a bilinear group $\mathbb{G}$ as follows

$$(p, q, N = pq, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot)) \leftarrow \mathsf{Gen}(1^\lambda).$$

It next samples random generators $g_p, h_p \in \mathbb{G}_p$ and $g_q, h_q \in \mathbb{G}_q$ and sets $g = g_p g_q$, $h = h_p h_q \in \mathbb{G}$. Additionally it chooses random exponents as follows

$$\begin{aligned} \forall\, \ell \in [\kappa], b \in \{0,1\}, &\qquad \delta_{\ell,b} \leftarrow \mathbb{Z}_N, \quad \gamma_{\ell,b} \leftarrow \mathbb{Z}_p \\ \forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, &\qquad c_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\, x \in [m], &\qquad r_x, \alpha_x \leftarrow \mathbb{Z}_N \end{aligned}$$

It also samples $\beta \leftarrow \mathbb{Z}_q$. Finally, it sets the master secret-public-tracing key tuple as follows

$$\mathsf{pk} = \left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \quad g, h, E_q = g_q^\beta, \\ \left\{ \begin{array}{c} E_x = g^{r_x},\ F_x = h^{r_x},\ G_x = e(g,g)^{\alpha_x}, \\ E_{q,x} = g_q^{\beta r_x},\ F_{q,x} = h_q^{\beta r_x},\ G_{q,x} = e(g_q,g_q)^{\beta\alpha_x} \end{array} \right\}_{x\in[m]}, \\ \{H_{y,\ell,b} = g^{c_{y,\ell,b}}\}_{(y,\ell,b)\in[\widetilde{m}]\times[\kappa]\times\{0,1\}}, \\ \left\{\widetilde{V}_{\ell,b} = g^{\delta_{\ell,b}} g_p^{\gamma_{\ell,b}},\ V_{\ell,b} = h^{\delta_{\ell,b}}\right\}_{(\ell,b)\in[\kappa]\times\{0,1\}} \end{array} \right),$$

$$\mathsf{msk} = \left(\begin{array}{c} g, \mathbb{G}, \quad \{r_x, \alpha_x\}_{x\in[m]}, \\ \{c_{y,\ell,b}\}_{(y,\ell,b)\in[\widetilde{m}]\times[\kappa]\times\{0,1\}} \end{array} \right), \qquad \mathsf{key} = \mathsf{pk}.$$

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa, i \in [n]) \to \mathsf{sk}$. The key generation algorithm first parses the key msk be as defined during setup, and also let $(x, y) \in [m] \times [\widetilde{m}]$ be the unique row-wise representation of index i. (That is, for any $i \in [n]$, its corresponding indices can be defined as $y = i \bmod \widetilde{m}$ and $x = \lceil i/\widetilde{m} \rceil$.)

It outputs the secret key as $\mathsf{sk} = (x, y, \mathsf{id}, g^{\alpha_x + r_x(\sum_{\ell\in[\kappa]} c_{y,\ell,\mathsf{id}_\ell})})$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The (normal) encryption algorithm is same as the special-encryption algorithm (described next) when run on index-position-value tuple $(i^*, \ell^*, b^*) = (1, \perp, 0)$. (Note that special-encryption is a public key algorithm since $\mathsf{key} = \mathsf{pk}$.)

[12] Again this is because the condition $(i > n+1) \vee (i = n+1 \wedge (\ell \neq \perp \vee b \neq b^*))$ is equivalent to $i \geq n+1$ which is always false.

[13] Note that since we are rounding up, thus the number of users that the system can support can be more than the input n provided. However, assymptotically the efficiency of the scheme is not affected.

$\mathsf{SplEnc}(\mathsf{key}, M \in \mathbb{G}_T, (i^*, \ell^*, b^*)) \to \mathsf{ct}$. The encryption algorithm first parses the key $\mathsf{key} = \mathsf{pk}$ be as defined during setup, and also let $(x^*, y^*) \in [m] \times [\widetilde{m}]$ be the unique row-wise representation of index i^*. It chooses random exponents as follows

$$\tau, t \in \mathbb{Z}_N, \qquad \forall\, x \in [m], \qquad s_x, e_x, f_x \leftarrow \mathbb{Z}_N$$
$$\forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, \qquad w_{y,\ell,b}, v_{y,\ell,b} \leftarrow \mathbb{Z}_N$$

Now the ciphertext ct consists of the following components

$$\mathsf{ct} = \begin{pmatrix} \left\{R_x, \widetilde{R}_x, A_x, B_x\right\}_{x \in [m]}, \\ \left\{C_{y,\ell,b}, \widetilde{C}_{y,\ell,b}\right\}_{(y,\ell,b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}} \end{pmatrix}$$

where each of the components are computed as described in Tables 2 and 3.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	${E_{q,x}}^{s_x}$	${F_{q,x}}^{s_x \tau}$	${E_q}^{s_x t}$	$M \cdot {G_{q,x}}^{s_x t}$
$= x^*$	${E_x}^{s_x}$	${F_x}^{s_x \tau}$	$g^{s_x t}$	$M \cdot {G_x}^{s_x t}$
$< x^*$	g^{s_x}	$h^{s_x \tau}$	g^{e_x}	$e(g,g)^{f_x}$

Table 2: Computing row components of the ciphertext for $x \in [m]$.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > y^*) \vee$ $(y = y^* \wedge (\ell, b) \neq (\ell^*, b^*))$	${H_{y,\ell,b}}^t \cdot h^{w_{y,\ell,b} \tau}$	$g^{w_{y,\ell,b}}$
$(y < y^*) \vee$ $(y, \ell, b) = (y^*, \ell^*, b^*)$	${H_{y,\ell,b}}^t \cdot h^{w_{y,\ell,b} \tau} \cdot {V_{\ell,b}}^{v_{y,\ell,b} \tau}$	$g^{w_{y,\ell,b}} \cdot \widetilde{V}_{\ell,b}^{v_{y,\ell,b}}$

Table 3: Computing column components of the ciphertext for $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to M \in \mathbb{G}_T$. The encryption algorithm first parses the ciphertext ct as defined during encryption. Also, let $\mathsf{sk} = (x, y, \mathsf{id}, K)$ where $K \in \mathbb{G}$. It then computes the following and outputs the message M.

$$M = \frac{B_x \cdot e(R_x, \prod_\ell C_{y,\ell,\mathsf{id}_\ell})}{e(\widetilde{R}_x, \prod_\ell \widetilde{C}_{y,\ell,\mathsf{id}_\ell}) \cdot e(K, A_x)}$$

Correctness. The proof of correctness is similar to that of the Boneh-Waters [BW06] scheme. Below we briefly highlight the main points.

Consider a secret key $\mathsf{sk} = (x, y, \mathsf{id}, K)$ for index $i = (x, y)$ and identity id. We know that $K = g^{\alpha_x} g^{r_x (\sum_{\ell \in [\kappa]} c_{y,\ell,\mathsf{id}_\ell})}$. Also, consider a ciphertext ct encrypting message M for index-position-value tuple (i^*, ℓ^*, b^*) where ciphertext ct consists of $\{R_x, \widetilde{R}_x, A_x, B_x\}_x, \{C_{y,\ell,b}, \widetilde{C}_{y,\ell,b}\}_{y,\ell,b}$. Recall that for correctness we require that the decryption algorithm outputs the message M if $i \geq i^* + 1$, or $(i^*, \ell^*) = (i, \perp)$, or $(i^*, \mathsf{id}_{\ell^*}) = (i, 1 - b^*)$. Let $i^* = (x^*, y^*)$. Suppose that index i and identity id satisfies these constraints. Now consider the following two cases:

Case 1: $x > x^*$. In this scenario, we have that the $R_x = {E_{q,x}}^{s_x}$, $\widetilde{R}_x = {F_{q,x}}^{s_x \tau}$, $A_x = {E_q}^{s_x t}$, $B_x = M \cdot {G_{q,x}}^{s_x t}$ (see Table 2). Now irrespective of the whether $y > y^*$ or not, we can simplify $e(R_x, C_{y,\ell,b})$ and $e(\widetilde{R}_x, \widetilde{C}_{y,\ell,b})$ as in Table 4.

Therefore, we get that for every $\ell \in [\kappa]$,

$$\frac{e(R_x, C_{y,\ell,\mathsf{id}_\ell})}{e(\widetilde{R}_x, \widetilde{C}_{y,\ell,\mathsf{id}_\ell})} = e(g_q, g_q)^{\beta r_x s_x c_{y,\ell,\mathsf{id}_\ell} t}.$$

	$e(R_x, C_{y,\ell,b})$	$e(\widetilde{R}_x, \widetilde{C}_{y,\ell,b})$
$(y > y^*) \vee$ $(y = y^* \wedge (\ell, b) \neq (\ell^*, b^*))$	$e(g_q, g_q)^{\beta r_x s_x c_{y,\ell,b} t}$ $\cdot e(g_q, h_q)^{\beta r_x s_x w_{y,\ell,b} \tau}$	$e(g_q, h_q)^{\beta r_x s_x w_{y,\ell,b} \tau}$
$(y < y^*) \vee$ $(y, \ell, b) = (y^*, \ell^*, b^*)$	$e(g_q, g_q)^{\beta r_x s_x c_{y,\ell,b} t}$ $\cdot e(g_q, h_q)^{\beta r_x s_x w_{y,\ell,b} \tau}$ $\cdot e(g_q, h_q)^{\beta r_x s_x \delta_{\ell,b} v_{y,\ell,b}}$	$e(g_q, h_q)^{\beta r_x s_x w_{y,\ell,b} \tau}$ $\cdot e(g_q, h_q)^{\beta r_x s_x \delta_{\ell,b} v_{y,\ell,b}}$

Table 4: Partial evaluations 1.

Also, we have that $e(K, A_x) = e(g_q, g_q)^{\beta \alpha_x} e(g_q, g_q)^{\beta r_x (\sum_\ell c_{y,\ell,\mathsf{id}_\ell})}$. Therefore, we get that

$$\begin{aligned}
\frac{B_x \cdot e(R_x, \prod_\ell C_{y,\ell,\mathsf{id}_\ell})}{e(\widetilde{R}_x, \prod_\ell \widetilde{C}_{y,\ell,\mathsf{id}_\ell}) \cdot e(K, A_x)} &= \frac{B_x \cdot \cancel{\prod_\ell e(g_q, g_q)^{\beta r_x s_x c_{y,\ell,\mathsf{id}_\ell} t}}}{e(g_q, g_q)^{\beta \alpha_x s_x t} \cdot \cancel{e(g_q, g_q)^{\beta r_x (\sum_\ell c_{y,\ell,\mathsf{id}_\ell}) s_x t}}} \\
&= \frac{B_x}{e(g_q, g_q)^{\beta \alpha_x s_x t}} = \frac{M \cdot \cancel{e(g_q, g_q)^{\beta \alpha_x s_x t}}}{\cancel{e(g_q, g_q)^{\beta \alpha_x s_x t}}} = M
\end{aligned}$$

Thus, correctness follows.

Case 2: (otherwise). Now in this scenario, we have that $R_x = E_x{}^{s_x}$, $\widetilde{R}_x = F_x{}^{s_x \tau}$, $A_x = g^{s_x t}$, $B_x = M \cdot G_x{}^{s_x t}$ (see Table 2). Since the correctness only needs to hold if $(i^*, \ell^*) = (i, \perp)$, or $(i^*, \mathsf{id}_{\ell^*}) = (i, 1 - b^*)$ (which is same as $(y > y^*) \vee (y = y^* \wedge (\ell, b) \neq (\ell^*, b^*))$), thus we can simplify $e(R_x, C_{y,\ell,b})$ and $e(\widetilde{R}_x, \widetilde{C}_{y,\ell,b})$ as in Table 5.

	$e(R_x, C_{y,\ell,b})$	$e(\widetilde{R}_x, \widetilde{C}_{y,\ell,b})$
$(y > y^*) \vee$ $(y = y^* \wedge (\ell, b) \neq (\ell^*, b^*))$	$e(g, g)^{\beta r_x s_x c_{y,\ell,b} t}$ $\cdot e(g, g)^{\beta r_x s_x w_{y,\ell,b} \tau}$	$e(g, g)^{\beta r_x s_x w_{y,\ell,b} \tau}$

Table 5: Partial evaluations 2.

And we can perform the rest of cancellations as in Case 1. Thus, correctness follows.

Efficiency. The EIPLBE scheme $\mathsf{EIPLBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec})$ is ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}\widetilde{e}}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)-efficient, where the efficiency measures are as follows:

- $\mathsf{T\text{-}s}(\lambda, \kappa, n) = \max(\widetilde{m} \cdot \kappa, m) \cdot \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}k}(\lambda, \kappa, n) = \max(\widetilde{m} \cdot \kappa, m) \cdot \mathsf{poly}(\lambda)$,[14]
- $\mathsf{T\text{-}e}(\lambda, \kappa, n) = \max(\widetilde{m} \cdot \kappa, m) \cdot \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}\widetilde{e}}(\lambda, \kappa, n) = \max(\widetilde{m} \cdot \kappa, m) \cdot \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}d}(\lambda, \kappa, n) = \max(\widetilde{m} \cdot \kappa, m) \cdot \mathsf{poly}(\lambda)$,[15]
- $\mathsf{S\text{-}c}(\lambda, \kappa, n) = \max(\widetilde{m} \cdot \kappa, m) \cdot \mathsf{poly}(\lambda)$,
- $\mathsf{S\text{-}k}(\lambda, \kappa, n) = \mathsf{poly}(\lambda)$.[16]

Note that $\max(\widetilde{m} \cdot \kappa, m)$ varies as follows.

$$\max(\widetilde{m} \cdot \kappa, m) = \begin{cases} \leq 2\sqrt{n \cdot \kappa} & \text{if } n \geq \kappa, \\ \kappa & \text{otherwise.} \end{cases}$$

[14]Here we include the time taken to parse the master secret key and select master secret key components as well.

[15]Here we include the time taken to parse the ciphertext and select ciphertext components as well.

[16]Here we ignore the fact that the secret key also contains the user index and identity. If we include that towards key size, then $\mathsf{S\text{-}k}$ increases by an additive amount of $\kappa + \log n$.

7.1 Security

In this section, we prove security of our construction. Formally, we prove the following.

Theorem 7.1. If the assumptions 1, 2, and 3 hold over the group generator Gen, then the scheme $\mathsf{EIPLBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec})$ described above is a secure EIPLBE as per Definitions 5.1 to 5.5.

We prove the above theorem in parts where we prove that our construction satisfies all five security properties. Parts of the security proof are similar to that of PLBE index hiding proof of [BW06]. Before we sketch the main ideas behind the proof, we would like to highlight that since the EIPLBE scheme described above has a public key special-encryption algorithm thus the adversary does not need to query the EIPLBE challenger for special-encryption queries since the adversary can simulate them on its own. Therefore, throughout this section we consider that the adversary queries the challenger for secret keys *only*.

7.1.1 Normal Hiding

Since the distributions of (normal) ciphertexts and (special-encryption) ciphertexts for index-position-value tuple $(1, \perp, 0)$ are the same, thus the normal hiding security of the scheme follows. (See construction.)

7.1.2 Index Hiding

Recall that the adversary in the index hiding game outputs the challenge index $i^* = (x^*, y^*)$ at the beginning (see Definition 5.2) where the adversary must not be able to distinguish between encryptions to $(i^*, \perp, 0)$ and $(i^* + 1, \perp, 0)$ if it does not have a secret key corresponding to index i^*. Now if $y^* = \widetilde{m}$, then we have that $i^* + 1 = (x^* + 1, 1)$. Otherwise $i^* + 1 = (x^*, y^* + 1)$. Similar to the [BW06] proof, we break down the proof in two parts based on whether $y^* = \widetilde{m}$ or not.

Case 1: $(y^* < \widetilde{m})$. In this scenario we need to show that an encryption to $(x^*, y^*, \perp, 0)$ and an encryption to $(x^*, y^* + 1, \perp, 0)$ is indistinguishable. Formally, we prove the following lemma.

Lemma 7.1. If the assumption 1 holds over the group generator Gen, then no polynomial time adversary can distinguish between an encryption to $(x^*, y^*, \perp, 0)$ and an encryption to $(x^*, y^* + 1, \perp, 0)$ with non-negligible advantage (where $i^* = (x^*, y^*)$ is the index output by the adversary in the index hiding game and the adversary is not allowed to make key query for index i^*).

Proof. The proof of this lemma follows via a sequence of $2\kappa + 1$ hybrid games $H_0, H_{\widetilde{\ell},\widetilde{b}}$ for $\widetilde{\ell} \in [\kappa]$ and $\widetilde{b} \in \{0, 1\}$. Here the hybrid H_0 corresponds to the index hiding game in which the challenge ciphertext is an encryption to index-position-value tuple $(i^* = (x^*, y^*), \perp, 0)$ where $y^* < \widetilde{m}$. And, the hybrid $H_{\widetilde{\ell},\widetilde{b}}$ is identical to hybrid H_0, except the column components in the challenge ciphertext $C_{y^*,\ell,b}$ for $(\ell, b) \in [\widetilde{\ell} - 1] \times \{0, 1\}$ and $\ell = \widetilde{\ell}, b \le \widetilde{b}$ have a random component in the $\mathbb{G}_p$ subgroup. Concretely, in the $H_{\widetilde{\ell},\widetilde{b}}$ hybrid the column component is computed as in Table 6.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > y^*) \vee$ $(y = y^* \wedge \ell > \widetilde{\ell}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b > \widetilde{b})$	${H_{y,\ell,b}}^t \cdot h^{w_{y,\ell,b}\tau}$	$g^{w_{y,\ell,b}}$
$(y < y^*) \vee$ $(y = y^* \wedge \ell < \widetilde{\ell}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b \le \widetilde{b})$	${H_{y,\ell,b}}^t \cdot h^{w_{y,\ell,b}\tau} \cdot {V_{\ell,b}}^{v_{y,\ell,b}\tau}$	$g^{w_{y,\ell,b}} \cdot \widetilde{V}_{\ell,b}^{v_{y,\ell,b}}$

Table 6: Computing column components of the ciphertext in Hybrid $H_{\widetilde{\ell},\widetilde{b}}$.

Note that the hybrid $H_{\ell,1}$ corresponds to the index hiding game in which the challenge ciphertext is an encryption to index-position-value tuple $(x^*, y^*+1, \perp, 0)$. Now to prove index hiding security it is sufficient to prove that hybrids $H_{\widetilde{\ell},\widetilde{b}}$ and $H_{\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}+1) \bmod 2}$ are indistinguishable for all $\widetilde{\ell}, \widetilde{b}$. (In addition, it is also required that hybrids H_0 and $H_{1,0}$ are indistinguishable.) Below we just show that hybrids $H_{\widetilde{\ell},\widetilde{b}}$ and $H_{\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}+1) \bmod 2}$ are indistinguishable as exactly same ideas could be applied for proving indistinguishability of remaining consecutive hybrids however proving them together seems to involve a heavy notational overhead. Now note that, combining these claims, the main lemma would follow.

Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$ that distinguishes hybrids $H_{\widetilde{\ell},\widetilde{b}}$ and $H_{\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}-1) \bmod 2}$ with non-negligible advantage $\epsilon(\cdot)$. We build a PPT reduction algorithm $\mathcal{B}$ that breaks assumption 1 with advantage $\epsilon(\cdot)$ as follows.

The reduction algorithm $\mathcal{B}$ first receives the modified DBDH challenge from the challenger as

$$(N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_q, g_p, A = g_p^a, B = g_p^b, C = g_p^c, D = g_p^{b^2}, T)$$

where T is either g_p^{abc} or a random group element in the subgroup of prime order p. Next, it receives the challenge tuple $(1^\kappa, 1^n, x^*, y^*)$ from the adversary $\mathcal{A}$ where $y^* < \widetilde{m}$.

Now $\mathcal{B}$ needs to generate the public key and send it to the adversary. After that adversary makes polynomially many key queries for distinct indices, and sends a challenge message m. The key queries and challenge message could be arbitrarily interleaved. Below we show that first how does $\mathcal{B}$ generates the public key and later describe how to answer key queries and output challenge ciphertext. Note that finally the adversary outputs its guess, which $\mathcal{B}$ uses to break assumption 1.

Main idea. Since the reduction plays the game with its challenger in the subgroup $\mathbb{G}_p$, thus it can choose everything in the $\mathbb{G}_q$ subgroup by itself. And the main idea is to implicitly set the exponents r_{p,x^*}, s_{p,x^*} as $b \cdot \widetilde{r}_{p,x^*}, \widetilde{s}_{p,x^*}/b$ (respectively) where $\widetilde{r}_{p,x^*}, \widetilde{s}_{p,x^*}$ are chosen at random, and also implicitly set $t_p = a \cdot b$, $h_p = B = g_p^b$ and $c_{p,y^*,\widetilde{\ell},\widetilde{b}} = c \cdot \widetilde{c}_{p,y^*,\widetilde{\ell},\widetilde{b}}$ where exponent $\widetilde{c}_{p,y^*,\widetilde{\ell},\widetilde{b}}$ is chosen at random. Setting exponents this way allows us to simulate the public key and secret keys exactly as well as the challenge group element T can be programmed in the challenge ciphertext component $C_{y^*,\widetilde{\ell},\widetilde{b}}$.

Public key. It chooses random generator $h_q \in \mathbb{G}_q$ by sampling random exponent $d \in \mathbb{Z}_N$ and setting $h_q = g_q^d$. Additionally it chooses random exponents as follows

$$\begin{aligned} \forall\ \ell \in [\kappa], b \in \{0,1\}, &\qquad \delta_{\ell,b} \leftarrow \mathbb{Z}_N, \quad \gamma_{\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, &\qquad \widetilde{c}_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ x \in [m], &\qquad \widetilde{r}_x \leftarrow \mathbb{Z}_N,\ \alpha_x \leftarrow \mathbb{Z}_N \end{aligned}$$

It also samples $\beta \leftarrow \mathbb{Z}_N$. Next it computes key components $\{E_x, F_x, G_x, E_{q,x}, F_{q,x}, G_{q,x}\}, \{H_{y,\ell,b}\}$ for $x \in [m]$ and $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$ as

$$E_x = \begin{cases} (g_p g_q)^{\widetilde{r}_x} & \text{for } x \neq x^*, \\ (B g_q)^{\widetilde{r}_x} & \text{otherwise.} \end{cases}, \qquad F_x = \begin{cases} (B h_q)^{\widetilde{r}_x} & \text{for } x \neq x^*, \\ (D h_q)^{\widetilde{r}_x} & \text{otherwise.} \end{cases},$$

$$H_{y,\ell,b} = \begin{cases} (C g_q)^{\widetilde{c}_{y,\ell,b}} & \text{for } y = y^*, \ell = \widetilde{\ell}, b = \widetilde{b}, \\ (g_p g_q)^{\widetilde{c}_{y,\ell,b}} & \text{otherwise.} \end{cases},$$

$$E_{q,x} = g_q^{\beta \widetilde{r}_x},\ F_{q,x} = h_q^{\beta \widetilde{r}_x},\ G_{q,x} = e(g_q, g_q)^{\beta \alpha_x},\ G_x = e(g,g)^{\alpha_x}.$$

Finally, it sets the public key as

$$\mathsf{pk} = \left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \quad g = g_p g_q, h = B h_q, E_q = g_q^\beta, \\ \left\{ \begin{array}{c} E_x,\ F_x,\ G_x, \\ E_{q,x},\ F_{q,x},\ G_{q,x} \end{array} \right\}_x, \{H_{y,\ell,b}\}_{y,\ell,b}, \\ \left\{ \widetilde{V}_{\ell,b} = g^{\delta_{\ell,b}} g_p^{\gamma_{\ell,b}},\ V_{\ell,b} = h^{\delta_{\ell,b}} \right\}_{\ell,b} \end{array} \right)$$

Note that all the above terms can be computed using only the modified DBDH challenge.

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x, y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = \begin{cases} g^{\alpha_x} g^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x \neq x^*, y \neq y^* \\ g^{\alpha_x} (B g_q)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x = x^*, (y \neq y^* \text{ or } \mathsf{id}_{\widetilde{\ell}} \neq \widetilde{b}) \\ g^{\alpha_x} g^{\widetilde{r}_x (\sum_{\ell \neq \widetilde{\ell}} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} (C g_q)^{\widetilde{r}_x \widetilde{c}_{y,\widetilde{\ell},\widetilde{b}}} & \text{if } x \neq x^*, y = y^*, \mathsf{id}_{\widetilde{\ell}} = \widetilde{b} \end{cases}$$

Note that the adversary is not allowed to query for index $i^* = (x^*, y^*)$.

Challenge ciphertext. It chooses random exponents as follows

$$\begin{array}{lll} \tau \in \mathbb{Z}_N, \ t_q \leftarrow \mathbb{Z}_N, & \forall\, x \in [m], & e_x, f_x \leftarrow \mathbb{Z}_N, \widetilde{s}_x \leftarrow \mathbb{Z}_N \\ & \forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0, 1\}, & \widetilde{w}_{y,\ell,b} \leftarrow \mathbb{Z}_N, v_{y,\ell,b} \leftarrow \mathbb{Z}_N \end{array}$$

Now the ciphertext components are computed as described in Tables 7 and 8.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	$E_{q,x}^{\widetilde{s}_x}$	$F_{q,x}^{\widetilde{s}_x \tau}$	$E_q^{\widetilde{s}_x t_q}$	$M \cdot G_{q,x}^{\widetilde{s}_x t_q}$
$= x^*$	$g^{\widetilde{r}_x \widetilde{s}_x}$	$(B h_q)^{\widetilde{r}_x \widetilde{s}_x \tau}$	$(A g_q^{t_q})^{\widetilde{s}_x}$	$M \cdot e(g, A g_q^{t_q})^{\alpha_x \widetilde{s}_x}$
$< x^*$	$g^{\widetilde{s}_x}$	$(B h_q)^{\widetilde{s}_x \tau}$	g^{e_x}	$e(g, g)^{f_x}$

Table 7: Computing row components of the ciphertext for $x \in [m]$.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > y^*) \vee$ $(y = y^* \wedge \ell > \widetilde{\ell}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b > \widetilde{b})$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau}$	$A^{-\widetilde{c}_{y,\ell,b}/\tau} g^{\widetilde{w}_{y,\ell,b}}$
$y = y^* \wedge \ell = \widetilde{\ell} \wedge b = \widetilde{b}$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau} T^{\widetilde{c}_{y,\ell,b}}$	$g^{\widetilde{w}_{y,\ell,b}}$
$(y < y^*) \vee$ $(y = y^* \wedge \ell < \widetilde{\ell}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b < \widetilde{b})$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau} g_p^{v_{y,\ell,b}}$	$g^{\widetilde{w}_{y,\ell,b}}$

Table 8: Computing column components of the ciphertext for $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0, 1\}$.

Finally, $\mathcal{B}$ receives a guess b' from $\mathcal{A}$ and it simply forwards that as its guess to the modified DBDH challenger.

Analysis. If $T = g_p^{abc}$, then $\mathcal{B}$ simulates the view of hybrid $H_{\widetilde{\ell},\widetilde{b}}$, otherwise if T is randomly chosen group element in $\mathbb{G}_p$ subgroup then $\mathcal{B}$ simulates the view of hybrid $H_{\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}-1) \bmod 2}$. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 1 with advantage ϵ.

□

Case 2: $(y^* = \widetilde{m})$**.** In this scenario we need to show that an encryption to $(x^*, y^* = \widetilde{m}, \perp, 0)$ and an encryption to $(x^* + 1, 1, \perp, 0)$ is indistinguishable. Formally, we prove the following lemma.

Lemma 7.2. If the assumptions 1, 2, 3, and 4 holds over the group generator Gen, then no polynomial time adversary can distinguish between an encryption to $(x^*, y^* = \widetilde{m}, \perp, 0)$ and an encryption to $(x^* + 1, 1, \perp, 0)$ with non-negligible advantage (where $i^* = (x^*, y^* = \widetilde{m})$ is the index output by the adversary in the index hiding game and the adversary is not allowed to make key query for index i^*).

Proof. The proof of this lemma follows by a sequence of hybrid experiments.

Hybrid 1. This hybrid corresponds to the index hiding game in which the challenge ciphertext is an encryption to index-position-value tuple $(i^* = (x^*, y^*), \perp, 0)$ where $y^* = \widetilde{m}$.

Hybrid 2. This hybrid is same as previous hybrid, except that the challenge ciphertext is an encryption to index-position-value tuple $(i^* = (x^*, y^* + 1), \perp, 0)$ where $y^* = \widetilde{m}$.[17] (see Table 3)

Hybrid 3. This hybrid is same as previous hybrid, except that the row components in the challenge ciphertext are computed as in Table 9 where $L = e(g_p, g)^z$ and $z \in \mathbb{Z}_p$ is a random exponent.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	${E_{q,x}}^{s_x}$	${F_{q,x}}^{s_x\tau}$	${E_q}^{s_x t}$	$M \cdot {G_{q,x}}^{s_x t}$
$= x^*$	${E_x}^{s_x}$	${F_x}^{s_x\tau}$	$g^{s_x t}$	$M \cdot {G_x}^{s_x t} \cdot L$
$< x^*$	g^{s_x}	$h^{s_x\tau}$	g^{e_x}	$e(g,g)^{f_x}$

Table 9: Computing row components of the ciphertext for $x \in [m]$ in Hybrids 3 and 4.

Hybrid 4. This hybrid is same as previous hybrid, except that the row components in the challenge ciphertext are computed as in Table 9 where $L = e(g, g)^z$ and $z \in \mathbb{Z}_N$ is a random exponent.

Hybrid 5. This hybrid is same as previous hybrid, except that the row components in the challenge ciphertext are computed as in Table 10.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	${E_{q,x}}^{s_x}$	${F_{q,x}}^{s_x\tau}$	${E_q}^{s_x t}$	$M \cdot {G_{q,x}}^{s_x t}$
$\leq x^*$	g^{s_x}	$h^{s_x\tau}$	g^{e_x}	$e(g,g)^{f_x}$

Table 10: Computing row components of the ciphertext for $x \in [m]$ in Hybrid 5.

Hybrid 6. This hybrid is same as previous hybrid, except that in the challenge ciphertext the column components are encrypted to $(y^* = 1, \ell^* = \perp, b^* = 0)$. (see Table 3. In words, in this hybrid the challenger encrypts to first index in the column components, and row x^* is less than row, $x^* + 1$ is greater than row.)

Hybrid 7. This hybrid corresponds to the index hiding game in which the challenge ciphertext is an encryption to index-position-value tuple $(i^* + 1 = (x^*, 1), \perp, 0)$ where $y^* = \widetilde{m}$.

Next, we show via a sequence of claims that the adversary's advantage in each pair of consecutive hybrids is negligibly close thereby completing the proof.

[17] Note that the special-encryption algorithm does not legally encrypt to position $(x^*, y^* + 1 = \widetilde{m} + 1)$, however the algorithm can be naturally extended to encrypt to such position.

Claim 7.1. If the assumption 1 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 1 and 2 with non-negligible advantage.

Proof. The proof of above claim is identical to that of Lemma 7.1. $\square$

Claim 7.2. If the assumption 1 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 2 and 3 with non-negligible advantage.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$ that distinguishes between Hybrids 2 and 3 with non-negligible advantage $\epsilon(\cdot)$ with constraint that $y^* = \widetilde{m}$. We build a PPT reduction algorithm $\mathcal{B}$ that breaks assumption 1 with advantage $\epsilon(\cdot)$ as follows.[18]

The reduction algorithm $\mathcal{B}$ first receives the DBDH challenge from the challenger as

$$(N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_q, g_p, A = g_p^a, B = g_p^b, C = g_p^c, T = e(g_p, g)^z)$$

where z is either abc or a random element in $\mathbb{Z}_N$. Next, it receives the challenge tuple $(1^\kappa, 1^n, x^*, y^*)$ from the adversary $\mathcal{A}$ where $y^* = \widetilde{m}$.

Now $\mathcal{B}$ needs to generate the public key and send it to the adversary. After that adversary makes polynomially many key queries for distinct indices, and sends a challenge message m. The key queries and challenge message could be arbitrarily interleaved. Below we show that first how does $\mathcal{B}$ generates the public key and later describe how to answer key queries and output challenge ciphertext. Note that finally the adversary outputs its guess, which $\mathcal{B}$ uses to break assumption 1.

Main idea. Since the reduction plays the game with its challenger in the subgroup $\mathbb{G}_p$, thus it can choose everything in the $\mathbb{G}_q$ subgroup by itself. And the main idea is to implicitly set the exponents $r_{p,x^*} = b, \alpha_{p,x^*} = a \cdot b$, and $t_p = c$. Additionally, set $c_{p,y,\ell,b} = \widetilde{c}_{p,y,\ell,b} - a$ for all y, ℓ, b, where exponents $\widetilde{c}_{p,y,\ell,b}$ are chosen at random. Setting exponents this way allows us to simulate the public key and secret keys exactly as well as the challenge group element T can be programmed in the challenge ciphertext component B_{x^*}.

Public key. It chooses a random exponent $d \in \mathbb{Z}_N$. Additionally it chooses random exponents as follows

$$\begin{aligned} \forall\ \ell \in [\kappa], b \in \{0,1\}, &\qquad \delta_{\ell,b} \leftarrow \mathbb{Z}_N, \quad \gamma_{\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, &\qquad \widetilde{c}_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ x \in [m], &\qquad \widetilde{r}_x \leftarrow \mathbb{Z}_N,\ \widetilde{\alpha}_x \leftarrow \mathbb{Z}_N \end{aligned}$$

It also samples $\beta \leftarrow \mathbb{Z}_N$. Next it computes key components $\{E_x, F_x, G_x, E_{q,x}, F_{q,x}, G_{q,x}\}, \{H_{y,\ell,b}\}$ for $x \in [m]$ and $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$ as

$$E_x = \begin{cases} (g_p g_q)^{\widetilde{r}_x} & \text{for } x \neq x^*, \\ B g_q^{\widetilde{r}_x} & \text{otherwise.} \end{cases}, \qquad F_x = \begin{cases} (g_p g_q)^{d\widetilde{r}_x} & \text{for } x \neq x^*, \\ B^d g_q^{d\widetilde{r}_x} & \text{otherwise.} \end{cases},$$

$$G_x = \begin{cases} e(g_p g_q, g_p g_q)^{\widetilde{\alpha}_x} & \text{for } x \neq x^*, \\ e(A, B)^\kappa e(g_q, g_q)^{\widetilde{\alpha}_x} & \text{otherwise.} \end{cases},$$

$$H_{y,\ell,b} = A^{-1}(g_p g_q)^{\widetilde{c}_{y,\ell,b}},\ E_{q,x} = g_q^{\beta\widetilde{r}_x},\ F_{q,x} = g_q^{d\beta\widetilde{r}_x},\ G_{q,x} = e(g_q, g_q)^{\beta\widetilde{\alpha}_x}.$$

Finally, it sets the public key as

$$\mathsf{pk} = \begin{pmatrix} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \quad g = g_p g_q, h = g^d, E_q = g_q^\beta, \\ \left\{ \begin{matrix} E_x,\ F_x,\ G_x, \\ E_{q,x},\ F_{q,x},\ G_{q,x} \end{matrix} \right\}_x, \{H_{y,\ell,b}\}_{y,\ell,b}, \\ \left\{ \widetilde{V}_{\ell,b} = g^{\delta_{\ell,b}} g_p^{\gamma_{\ell,b}},\ V_{\ell,b} = h^{\delta_{\ell,b}} \right\}_{\ell,b} \end{pmatrix}$$

Note that all the above terms can be computed using only the DBDH challenge.

[18]Technically, we only rely on Decisional Bilinear Diffie-Hellman (DBDH) assumption here.

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x, y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = \begin{cases} g^{\widetilde{\alpha}_x} A^{-\kappa \widetilde{r}_x} g^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x \neq x^*, \\ g_q^{\widetilde{\alpha}_x} B^{\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell}} g_q^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} c_{y,\ell,\mathsf{id}_\ell})} & \text{otherwise.} \end{cases}$$

Note that the adversary is not allowed to query for index $i^* = (x^*, y^*)$.

Challenge ciphertext. $\mathcal{B}$ can compute all the column components on its own since the $\mathbb{G}_p$ subgroup components are random in $C_{y,\ell,b}, \widetilde{C}_{y,\ell,b}$ terms, and for computing the $\mathbb{G}_q$ subgroup components $\mathcal{B}$ already knows all the required exponents. Similarly, all the row components $R_x, \widetilde{R}_x, A_x, B_x$ values for $x < x^*$ are just created randomly, and for $x > x^*$ they can be created by $\mathcal{B}$'s knowledge of all required exponents since they are only drawn from the $\mathbb{G}_q$ subgroup which it knows. Now for computing row components $R_{x^*}, \widetilde{R}_{x^*}, A_{x^*}, B_{x^*}$, the algorithm $\mathcal{B}$ proceeds as follows

$$R_{x^*} = (B\ g_q^{\widetilde{r}_{x^*}})^{\widetilde{s}_{x^*}}, \quad \widetilde{R}_{x^*} = (B\ g_q^{\widetilde{r}_{x^*}})^{d \widetilde{s}_{x^*} \tau}, \quad A_{x^*} = (C g_q^{t_q})^{\widetilde{s}_{x^*}}, \quad B_{x^*} = M \cdot T^{\kappa \widetilde{s}_{x^*}} e(g_q, g_q)^{\widetilde{\alpha}_{x^*} \widetilde{s}_{x^*} t_q}$$

where exponents $\widetilde{s}_{x^*}, \tau \in \mathbb{Z}_N$ and $t_q \leftarrow \mathbb{Z}_N$ are sampled uniformly at random. Finally, $\mathcal{B}$ receives a guess b' from $\mathcal{A}$ and it simply forwards that as its guess to the DBDH challenger.

Analysis. If $T = e(g_p, g)^{abc}$, then $\mathcal{B}$ simulates the view of hybrid 2, otherwise if $T = e(g_p, g)^z$ is randomly chosen group element in $\mathbb{G}_{T,p}$ subgroup then $\mathcal{B}$ simulates the view of hybrid 3. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 1 with advantage ϵ. □

Claim 7.3. If the assumption 3 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 3 and 4 with non-negligible advantage.

Proof. The proof of this lemma is identical to the indistinguishability proof of Hybrids $2a$ and $2b$ in [BW06, Section C.2, Claim 5.5]. Here we sktech the main ideas. The reduction algorithm $\mathcal{B}$ will first receive the Bilinear Subgroup Decision (BSD) challenge from the challenger which consists of $N, \mathbb{G}, \mathbb{G}_T, e(\cdot, \cdot), g_p, g_q, e(T, g)$ where T is either a random group element in group $\mathbb{G}$, or is a random group element in the subgroup $\mathbb{G}_p$. Since $\mathcal{B}$ already has generators g_p and g_q, it can perform the setup honestly and send the public key to the adversary. Now it can also answer each key query from the adversary honestly. (Note the key queries could be arbitrarily interleaved.) Finally, in order to compute the challenge ciphertext, $\mathcal{B}$ computes all ciphertext components except B_{x^*} honestly. And, it simply sets $B_{x^*} = M \cdot G_{x^*}{}^{s_{x^*} t} \cdot e(T, g)$ where $e(T, g)$ is taken from the BSD challenge. It is easy to check that $\mathcal{B}$ perfectly simulates hybrids 3 and 4 for $\mathcal{A}$ depending upon how T is sampled. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 3 with advantage ϵ. □

Claim 7.4. If the assumption 4 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 4 and 5 with non-negligible advantage.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$ that distinguishes between Hybrids 4 and 5 with non-negligible advantage $\epsilon(\cdot)$ with constraint that $y^* = \widetilde{m}$. We build a PPT reduction algorithm $\mathcal{B}$ that breaks assumption 4 with advantage $\epsilon(\cdot)$ as follows.

The reduction algorithm $\mathcal{B}$ first receives the bilinear challenge from the challenger as

$$(N, \mathbb{G}, \mathbb{G}_T, e(\cdot, \cdot), I_p \in \mathbb{G}_p, I_q \in \mathbb{G}_q, A = I_q^a, B = I_p^{\widetilde{a}} I_q^{a^2}, C = I_p^{\widetilde{c}} I_q^c, T)$$

where T is either $I_q^{a^2 c}$ or a random group element in the subgroup $\mathbb{G}_q$. Next, it receives the challenge tuple $(1^\kappa, 1^n, x^*, y^*)$ from the adversary $\mathcal{A}$ where $y^* = \widetilde{m}$.

Now $\mathcal{B}$ needs to generate the public key and send it to the adversary. After that adversary makes polynomially many key queries for distinct indices, and sends a challenge message m. The key queries and challenge message could be arbitrarily interleaved. Below we show that first how does $\mathcal{B}$ generates the public key and later describe how to answer key queries and output challenge ciphertext. Note that finally the adversary outputs its guess, which $\mathcal{B}$ uses to break assumption 4.

Main idea. Since the reduction plays the game with its challenger *mostly* in the subgroup $\mathbb{G}_q$, thus it can choose most components in the $\mathbb{G}_p$ subgroup by itself. By most we mean that since B and C terms in the challenge have components in the $\mathbb{G}_p$ subgroup, thus some $\mathbb{G}_p$ exponents will also implicitly depend on $\widetilde{a}$ and $\widetilde{c}$ terms.

Now the main idea is to implicitly set $g_p = I_p$, $g_q = A$, $r_{q,x^*} = \widetilde{r}_{q,x^*}/a$, $r_{p,x^*} = \widetilde{r}_{p,x^*}$, $s_{q,x^*} = c$, $s_{p,x^*} = \widetilde{c}$, $t_q = a$, and $t_p = \widetilde{a}$. Here the $\widetilde{r}_{p,x^*}, \widetilde{r}_{q,x^*}$ terms are sampled jointly and randomly. Additionally, the reduction algorithm samples the remaining exponents like β, $\delta_{\ell,b}, \gamma_{\ell,b}$, $c_{y,\ell,b}$ etc. Note that at any point we do not sample the $\mathbb{G}_p$ and $\mathbb{G}_q$ exponents separately, but instead sample an exponent directly from $\mathbb{Z}_N$ and make sure that the distributions are not affected. (This is important because the reduction algorithm does not know the factorization.) Finally, setting exponents this way allows us to simulate the public key and secret keys exactly as well as the challenge group element T can be programmed in the challenge ciphertext component A_{x^*}.

Public key. It chooses random exponents as follows

$$\begin{aligned} & d, \beta \leftarrow \mathbb{Z}_N \\ \forall\, \ell \in [\kappa], b \in \{0,1\}, \quad & \delta_{\ell,b}, \gamma_{\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, \quad & c_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\, x \in [m], \quad & \widetilde{r}_x, \alpha_x \leftarrow \mathbb{Z}_N \end{aligned}$$

Next it computes key components $\{E_x, F_x, G_x, E_{q,x}, F_{q,x}, G_{q,x}\}, \{H_{y,\ell,b}\}$ for $x \in [m]$ and $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$ as

$$E_x = \begin{cases} (I_p A)^{\widetilde{r}_x} & \text{for } x \neq x^*, \\ (I_p I_q)^{\widetilde{r}_x} & \text{otherwise.} \end{cases}, \qquad E_{q,x} = \begin{cases} A^{\beta \widetilde{r}_x} & \text{for } x \neq x^*, \\ I_q^{\beta \widetilde{r}_x} & \text{otherwise.} \end{cases},$$

$$F_x = {E_x}^d, \; F_{q,x} = {E_{q,x}}^d, \; G_x = e(I_p A, I_p A)^{\alpha_x}, \; G_{q,x} = e(A,A)^{\beta \alpha_x}, \; H_{y,\ell,b} = (I_p A)^{c_{y,\ell,b}}.$$

Finally, it sets the public key as

$$\mathsf{pk} = \left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \quad g = I_p A, h = g^d, E_q = A^\beta, \\ \left\{ \begin{array}{c} E_x, \; F_x, \; G_x, \\ E_{q,x}, \; F_{q,x}, \; G_{q,x} \end{array} \right\}_x, \{H_{y,\ell,b}\}_{y,\ell,b}, \\ \left\{ \widetilde{V}_{\ell,b} = g^{\delta_{\ell,b}} g_p^{\gamma_{\ell,b}}, \; V_{\ell,b} = h^{\delta_{\ell,b}} \right\}_{\ell,b} \end{array} \right)$$

Note that all the above terms can be computed using only the bilinear challenge.

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x,y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = g^{\alpha_x} {E_x}^{\sum_{\ell \in [\kappa]} c_{y,\ell,\mathsf{id}_\ell}} = \begin{cases} g^{\alpha_x} (I_p A)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} c_{y,\ell,\mathsf{id}_\ell})} & \text{if } x \neq x^*, \\ g^{\alpha_x} (I_p I_q)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} c_{y,\ell,\mathsf{id}_\ell})} & \text{otherwise.} \end{cases}$$

Note that the adversary is not allowed to query for index $i^* = (x^*, y^*)$.

Challenge ciphertext. It chooses random exponents as follows

$$\begin{aligned} & \tau, \widetilde{t}_p \in \mathbb{Z}_N \\ \forall\, x \in [m], \quad & e_x, f_x, \widetilde{s}_x \leftarrow \mathbb{Z}_N \\ \forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, \quad & \widetilde{w}_{y,\ell,b} \leftarrow \mathbb{Z}_N, \widetilde{v}_{y,\ell,b} \leftarrow \mathbb{Z}_p \end{aligned}$$

Now the ciphertext components are computed as described in Tables 11 and 12.

Finally, $\mathcal{B}$ receives a guess b' from $\mathcal{A}$ and it simply forwards that as its guess to the bilinear challenger.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	$I_q^{\beta\widetilde{r}_x\widetilde{s}_x}$	$I_q^{\beta\widetilde{r}_x\widetilde{s}_x\tau d}$	$A^{\beta\widetilde{s}_x}$	$M \cdot e(A,A)^{\beta\widetilde{s}_x\alpha_x}$
$= x^*$	$C^{\widetilde{r}_x}$	$C^{\widetilde{r}_x\tau d}$	$I_p^{\widetilde{t}_p}T$	$e(g,g)^{f_x}$
$< x^*$	$g^{\widetilde{s}_x}$	$g^{\widetilde{s}_x\tau d}$	g^{e_x}	$e(g,g)^{f_x}$

Table 11: Computing row components of the ciphertext for $x \in [m]$.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$\forall\ y$	$B^{c_{y,\ell,b}} h^{\widetilde{w}_{y,\ell,b}\tau} I_p^{\widetilde{v}_{y,\ell,b}}$	$g^{\widetilde{w}_{y,\ell,b}}$

Table 12: Computing column components of the ciphertext for $(y,\ell,b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$.

Analysis. If $T = I_q^{a^2c}$, then $\mathcal{B}$ simulates the view of hybrid 4, otherwise if T is randomly chosen group element in $\mathbb{G}_q$ subgroup then $\mathcal{B}$ simulates the view of hybrid 5 as the target row '$= x^*$' will be statistically indistinguishable from the less than row '$< x^*$'. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 4 with advantage ϵ. □

Claim 7.5. If the assumption 1 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 5 and 6 with non-negligible advantage.

Proof. The proof of this lemma is similar to that of Lemma 7.1. The proof follows via a sequence of $2\widetilde{m}\kappa + 1$ sub-hybrid games $H_0, H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ for $\widetilde{y} \in [\widetilde{m}]$, $\widetilde{\ell} \in [\kappa]$ and $\widetilde{b} \in \{0,1\}$. Here the sub-hybrid H_0 corresponds to the main hybrid 5 described above. And, the hybrid $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ is identical to hybrid H_0, except the column components in the challenge ciphertext $C_{y,\ell,b}$ for $y < \widetilde{y}$ and $(y,\ell,b) \in \{\widetilde{y}\} \times [\widetilde{\ell}-1] \times \{0,1\}$ and $y = \widetilde{y}, \ell = \widetilde{\ell}, b < \widetilde{b}$ have a random component in the $\mathbb{G}_p$ subgroup. Concretely, in the $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ hybrid the column component is computed as in Table 15

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > \widetilde{y}) \vee$ $(y = \widetilde{y} \wedge \ell > \widetilde{\ell}) \vee$ $(y = \widetilde{y} \wedge \ell = \widetilde{\ell} \wedge b \geq \widetilde{b})$	${H_{y,\ell,b}}^{t} \cdot h^{w_{y,\ell,b}\tau}$	$g^{w_{y,\ell,b}}$
$(y < \widetilde{y}) \vee$ $(y = \widetilde{y} \wedge \ell < \widetilde{\ell}) \vee$ $(y = \widetilde{y} \wedge \ell = \widetilde{\ell} \wedge b < \widetilde{b})$	${H_{y,\ell,b}}^{t} \cdot h^{w_{y,\ell,b}\tau} \cdot {V_{\ell,b}}^{v_{y,\ell,b}\tau}$	$g^{w_{y,\ell,b}} \cdot \widetilde{V}_{\ell,b}^{v_{y,\ell,b}}$

Table 13: Computing column components of the ciphertext in Hybrid $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$.

Note that the sub-hybrid $H_{1,1,0}$ is same as main hybrid 6. Now to prove indistinguishability of main hybrids 5 and 6 it is sufficient to prove that sub-hybrids $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ and $H_{\widetilde{y},\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}+1) \bmod 2}$ are indistinguishable for all $\widetilde{y}, \widetilde{\ell}, \widetilde{b}$. (In addition, it is also required that sub-hybrids H_0 and $H_{\widetilde{m},\ell,1}$ are indistinguishable, as well as sub-hybrids $H_{\widetilde{y},\kappa,1}$ and $H_{\widetilde{y}+1,1,0}$.) Below we just show that hybrids $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ and $H_{\widetilde{y},\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}+1) \bmod 2}$ are indistinguishable as exactly same ideas could be applied for proving indistinguishability of remaining consecutive sub-hybrids however proving them together seems to involve a heavy notational overhead. Now note that, combining these sub-claims, the main claim would follow.

Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$ that distinguishes hybrids $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ and $H_{\widetilde{y},\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}-1) \bmod 2}$ with non-negligible advantage $\epsilon(\cdot)$. We build a PPT reduction algorithm $\mathcal{B}$ that breaks assumption 1 with advantage $\epsilon(\cdot)$ as follows.

The reduction algorithm $\mathcal{B}$ first receives the DBDH challenge from the challenger as

$$(N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_q, g_p, A = g_p^a, B = g_p^b, C = g_p^c, T)$$

where T is either g_p^{abc} or a random group element in the subgroup of prime order p. Next, it receives the challenge tuple $(1^\kappa, 1^n, x^*, y^*)$ from the adversary $\mathcal{A}$ where $y^* = \widetilde{m}$.

Now $\mathcal{B}$ needs to generate the public key and send it to the adversary. After that adversary makes polynomially many key queries for distinct indices, and sends a challenge message m. The key queries and challenge message could be arbitrarily interleaved. Below we show that first how does $\mathcal{B}$ generates the public key and later describe how to answer key queries and output challenge ciphertext. Note that finally the adversary outputs its guess, which $\mathcal{B}$ uses to break assumption 1.

Main idea. Since the reduction plays the game with its challenger in the subgroup $\mathbb{G}_p$, thus it can choose everything in the $\mathbb{G}_q$ subgroup by itself. And the main idea is to implicitly set the exponents $t_p = a \cdot b$, $h_p = B = g_p^b$ and $c_{p,\widetilde{y},\widetilde{\ell},\widetilde{b}} = c \cdot \widetilde{c}_{p,\widetilde{y},\widetilde{\ell},\widetilde{b}}$ where exponent $\widetilde{c}_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$ is chosen at random. Setting exponents this way allows us to simulate the public key and secret keys exactly as well as the challenge group element T can be programmed in the challenge ciphertext component $C_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$.

Public key. It chooses random generator $h_q \in \mathbb{G}_q$ by sampling random exponent $d \in \mathbb{Z}_N$ and setting $h_q = g_q^d$. Additionally it chooses random exponents as follows

$$\begin{aligned} \forall\ \ell \in [\kappa], b \in \{0,1\}, \quad & \delta_{\ell,b} \leftarrow \mathbb{Z}_N, \quad \gamma_{\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, \quad & \widetilde{c}_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ x \in [m], \quad & r_x \leftarrow \mathbb{Z}_N,\ \alpha_x \leftarrow \mathbb{Z}_N \end{aligned}$$

It also samples $\beta \leftarrow \mathbb{Z}_N$. It sets group elements $g = g_p g_q$, $h = B h_q$, and $E_q = g_q^\beta$. Next it computes key components $\{E_x, F_x, G_x, E_{q,x}, F_{q,x}, G_{q,x}\}, \{H_{y,\ell,b}\}$ for $x \in [m]$ and $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$ as

$$E_x = g^{r_x},\ F_x = h^{r_x},\ E_{q,x} = g_q^{\beta r_x},\ F_{q,x} = h_q^{\beta r_x},\ G_{q,x} = e(g_q, g_q)^{\beta\alpha_x},\ G_x = e(g,g)^{\alpha_x},$$

$$H_{y,\ell,b} = \begin{cases} (C g_q)^{\widetilde{c}_{y,\ell,b}} & \text{for } y = \widetilde{y}, \ell = \widetilde{\ell}, b = \widetilde{b}, \\ (g_p g_q)^{\widetilde{c}_{y,\ell,b}} & \text{otherwise.} \end{cases}.$$

Finally, it sets the public key as

$$\mathsf{pk} = \begin{pmatrix} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \quad g = g_p g_q, h = B h_q, E_q = g_q^\beta, \\ \left\{ \begin{array}{c} E_x,\ F_x,\ G_x, \\ E_{q,x},\ F_{q,x},\ G_{q,x} \end{array} \right\}_x, \{H_{y,\ell,b}\}_{y,\ell,b}, \\ \left\{ \widetilde{V}_{\ell,b} = g^{\delta_{\ell,b}} g_p^{\gamma_{\ell,b}},\ V_{\ell,b} = h^{\delta_{\ell,b}} \right\}_{\ell,b} \end{pmatrix}$$

Note that all the above terms can be computed using only the DBDH challenge.

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x,y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = \left(\prod_\ell H_{y,\ell,\mathsf{id}_\ell}\right)^{r_x} = \begin{cases} g^{\alpha_x} g^{r_x(\sum_{\ell \neq \widetilde{\ell}} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} (C g_q)^{r_x \widetilde{c}_{y,\widetilde{\ell},\widetilde{b}}} & \text{if } y = \widetilde{y},\ \mathsf{id}_{\widetilde{\ell}} = \widetilde{b} \\ g^{\alpha_x} g^{r_x(\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{otherwise.} \end{cases}$$

Note that the adversary is not allowed to query for index $i^* = (x^*, y^*)$.

Challenge ciphertext. It chooses random exponents as follows

$$\begin{aligned} \tau \in \mathbb{Z}_N,\ t_q \leftarrow \mathbb{Z}_N, \qquad \forall\ x \in [m], \quad & e_x, f_x \leftarrow \mathbb{Z}_N, s_x \leftarrow \mathbb{Z}_N \\ \forall\ y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, \quad & \widetilde{w}_{y,\ell,b} \leftarrow \mathbb{Z}_N, v_{y,\ell,b} \leftarrow \mathbb{Z}_N \end{aligned}$$

Now the ciphertext components are computed as described in Tables 14 and 15.

Finally, $\mathcal{B}$ receives a guess b' from $\mathcal{A}$ and it simply forwards that as its guess to the DBDH challenger.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	${E_{q,x}}^{s_x}$	${F_{q,x}}^{s_x \tau}$	${E_q}^{s_x t_q}$	$M \cdot {G_{q,x}}^{s_x t_q}$
$\leq x^*$	g^{s_x}	$h^{s_x \tau}$	g^{e_x}	$e(g,g)^{f_x}$

Table 14: Computing row components of the ciphertext for $x \in [m]$.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > \widetilde{y}) \vee$ $(y = \widetilde{y} \wedge \ell > \widetilde{\ell}) \vee$ $(y = \widetilde{y} \wedge \ell = \widetilde{\ell} \wedge b > \widetilde{b})$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau}$	$A^{-\widetilde{c}_{y,\ell,b}/\tau} g^{\widetilde{w}_{y,\ell,b}}$
$y = \widetilde{y} \wedge \ell = \widetilde{\ell} \wedge b = \widetilde{b}$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau} T^{\widetilde{c}_{y,\ell,b}}$	$g^{\widetilde{w}_{y,\ell,b}}$
$(y < \widetilde{y}) \vee$ $(y = \widetilde{y} \wedge \ell < \widetilde{\ell}) \vee$ $(y = \widetilde{y} \wedge \ell = \widetilde{\ell} \wedge b < \widetilde{b})$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau} g_p^{v_{y,\ell,b}}$	$g^{\widetilde{w}_{y,\ell,b}}$

Table 15: Computing column components of the ciphertext for $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$.

Analysis. If $T = g_p^{abc}$, then $\mathcal{B}$ simulates the view of hybrid $H_{\widetilde{y},\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}-1) \bmod 2}$, otherwise if T is randomly chosen group element in $\mathbb{G}_p$ subgroup then $\mathcal{B}$ simulates the view of hybrid $H_{\widetilde{y},\widetilde{\ell},\widetilde{b}}$. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 1 with advantage ϵ. □

Claim 7.6. If the assumption 2 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 6 and 7 with non-negligible advantage.

Proof. The proof of this lemma is similar to that of [BW06, Claim 5.7]. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$ that distinguishes hybrids 6 and 7 with non-negligible advantage $\epsilon(\cdot)$. We build a PPT reduction algorithm $\mathcal{B}$ that breaks assumption 2 with advantage $\epsilon(\cdot)$ as follows.

The reduction algorithm $\mathcal{B}$ first receives the DHSD challenge from the challenger as

$$(N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g = g_p g_q, h = h_p h_q, A = g_q^a, B = h_q^a, C = g^b g_p^c, D = h^b, T)$$

where T is either sampled as $T = g_q^d$ or $T = g^d$, where d is a random exponent sampled as $d \leftarrow \mathbb{Z}_N$. Next, it receives the challenge tuple $(1^\kappa, 1^n, x^*, y^*)$ from the adversary $\mathcal{A}$ where $y^* = \widetilde{m}$.

Now $\mathcal{B}$ needs to generate the public key and send it to the adversary. After that adversary makes polynomially many key queries for distinct indices, and sends a challenge message m. The key queries and challenge message could be arbitrarily interleaved. Below we show that first how does $\mathcal{B}$ generates the public key and later describe how to answer key queries and output challenge ciphertext. Note that finally the adversary outputs its guess, which $\mathcal{B}$ uses to break assumption 2.

Main idea. First note that the adversary neither knows the factorization of N, but it does get random generators of the subgroup $\mathbb{G}_q$. Since all the group elements the reduction algorithm has to output either lie in the full group, or they lie in the subgroup $\mathbb{G}_q$ but with a fixed additional exponent β, thus it implicitly sets $\beta = a$ and randomly simulates all other components. Concretely, the main idea is to implicitly set the exponents $\beta = a$, $s_{x^*+1} = d \cdot \widetilde{s}_{x^*+1}$, and $\gamma_{\ell,b} = c \cdot \widetilde{\gamma}_{\ell,b}$, and $\delta_{\ell,b} = \widetilde{\delta}_{\ell,b} + b \cdot \widetilde{\gamma}_{\ell,b}$ where exponents $\widetilde{\gamma}_{\ell,b}, \widetilde{\delta}_{\ell,b}$ are chosen at random. Additionally, $\mathcal{B}$ implicitly sets τ such that $h^\tau = g^{\widetilde{\tau}}$ where exponent $\widetilde{\tau}$ is sampled uniformly at random. Setting exponents this way allows us to simulate the public key and secret keys exactly as well as the challenge group element T can be programmed in the challenge ciphertext to compute the row components for $x = x^* + 1$.

Public key. It chooses random exponents as follows

$$\begin{aligned} \forall\ \ell \in [\kappa], b \in \{0,1\}, &\qquad \widetilde{\delta}_{\ell,b} \leftarrow \mathbb{Z}_N, \quad \widetilde{\gamma}_{\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, &\qquad c_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\ x \in [m], &\qquad r_x \leftarrow \mathbb{Z}_N,\ \alpha_x \leftarrow \mathbb{Z}_N \end{aligned}$$

It computes key components $\left\{\widetilde{V}_{\ell,b},\ V_{\ell,b}\right\}_{\ell,b}$ for $(\ell, b) \in [\kappa] \times \{0,1\}$ as

$$\widetilde{V}_{\ell,b} = g^{\widetilde{\delta}_{\ell,b}} C^{\widetilde{\gamma}_{\ell,b}}, \qquad V_{\ell,b} = h^{\widetilde{\delta}_{\ell,b}} D^{\widetilde{\gamma}_{\ell,b}}.$$

Finally, it sets the public key as

$$\mathsf{pk} = \left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \qquad g, h, E_q = A, \\ \left\{ \begin{array}{c} E_x = g^{r_x},\ F_x = h^{r_x},\ G_x = e(g,g)^{\alpha_x}, \\ E_{q,x} = A^{r_x},\ F_{q,x} = B^{r_x},\ G_{q,x} = e(A,g)^{\alpha_x} \end{array} \right\}_x, \{H_{y,\ell,b} = g^{c_{y,\ell,b}}\}_{y,\ell,b}, \\ \left\{\widetilde{V}_{\ell,b},\ V_{\ell,b}\right\}_{\ell,b} \end{array} \right)$$

Note that all the above terms can be computed using only the DHSD challenge.

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x,y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = \left(\prod_{\ell} H_{y,\ell,\mathsf{id}_\ell} \right)^{r_x}.$$

Note that the adversary is not allowed to query for index $i^* = (x^*, y^*)$.

Challenge ciphertext. It chooses random exponents as follows

$$\begin{aligned} \widetilde{\tau} \in \mathbb{Z}_N,\ t \leftarrow \mathbb{Z}_N, \qquad\qquad \forall\ x \in [m], &\qquad e_x, f_x, \widetilde{s}_x \leftarrow \mathbb{Z}_N \\ \forall\ y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, &\qquad w_{y,\ell,b} \leftarrow \mathbb{Z}_N, v_{y,\ell,b} \leftarrow \mathbb{Z}_N \end{aligned}$$

Now the ciphertext components are computed as described in Tables 16 and 17.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^* + 1$	$E_{q,x}{}^{\widetilde{s}_x}$	$E_{q,x}{}^{\widetilde{s}_x \widetilde{\tau}}$	$E_q{}^{\widetilde{s}_x t}$	$M \cdot G_{q,x}{}^{\widetilde{s}_x t}$
$= x^* + 1$	$T^{r_x \widetilde{s}_x}$	$T^{r_x \widetilde{s}_x \widetilde{\tau}}$	$T^{\widetilde{s}_x t}$	$M \cdot e(T,g)^{\alpha_x \widetilde{s}_x t}$
$\le x^*$	$g^{\widetilde{s}_x}$	$h^{\widetilde{s}_x \tau}$	g^{e_x}	$e(g,g)^{f_x}$

Table 16: Computing row components of the ciphertext for $x \in [m]$.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$\forall\ y, \ell, b$	$H_{y,\ell,b}{}^t g^{w_{y,\ell,b} \widetilde{\tau}}$	$g^{w_{y,\ell,b}}$

Table 17: Computing column components of the ciphertext for $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$.

Finally, $\mathcal{B}$ receives a guess b' from $\mathcal{A}$ and it simply forwards that as its guess to the DHSD challenger.

Analysis. If $T = g_q^d$, then $\mathcal{B}$ simulates the view of hybrid 6, otherwise if $T = g^d$ that is it is randomly chosen group element in $\mathbb{G}$ then $\mathcal{B}$ simulates the view of hybrid 7. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 2 with advantage ϵ. □

This concludes the proof of Lemma 7.2. □

This concludes the proof of index hiding.

7.1.3 Lower Identity Hiding

In this scenario we need to show that an encryption to (i^*, ℓ^*, b^*) and an encryption to $(i^*, \perp, 0)$ is indistinguishable. Formally, we prove the following lemma.

Lemma 7.3. If the assumption 1 holds over the group generator Gen, then no polynomial time adversary can distinguish between an encryption to (i^*, ℓ^*, b^*) and an encryption to $(i^*, \perp, 0)$ with non-negligible advantage (where $i^* = (x^*, y^*)$ is the index output by the adversary in the lower identity hiding game and the adversary is not allowed to make key query for index i^* with identity id such that $\mathsf{id}_{\ell^*} = b^*$).

Proof. The proof of this lemma is nearly identical to that of Lemma 7.1, except that it does not require any intermediate hybrids. But instead it follows directly follows from the indistinguishability of intermediate hybrids used in the proof of Lemma 7.1. Concretely, let $(i^* = (x^*, y^*), \ell^*, b^*)$ be the challenge tuple output by the adversary $\mathcal{A}$. Then, the hybrid H_{ℓ^*, b^*} (as defined in Lemma 7.1) corresponds exactly to the lower identity hiding game in which the challenge ciphertext is an encryption for index-position-value tuple (i^*, ℓ^*, b^*), and similarly H_{ℓ^*, b^*} corresponds to the lower identity hiding game in which the challenge ciphertext is an encryption for index-position-value tuple $(i^*, \perp, 0)$. Now the proof of indistinguishability is same as described in Lemma 7.1, however we need to assure that the reduction algorithm $\mathcal{B}$ could answer all key queries permissible as per the lower index hiding game. Below we highlight why the reduction algorithm (as described in Lemma 7.1) can answer all key queries of the form (i, id) where either $i \neq i^*$ or $\mathsf{id}_{\ell^*} \neq b^*$. (Below is how key queries were answered.)

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x, y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = \begin{cases} g^{\alpha_x} g^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x \neq x^*, y \neq y^* \\ g^{\alpha_x} (B g_q)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x = x^*, (y \neq y^* \text{ or } \mathsf{id}_{\ell^*} \neq b^*) \\ g^{\alpha_x} g^{\widetilde{r}_x (\sum_{\ell \neq \ell^*} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} (C g_q)^{\widetilde{r}_x \widetilde{c}_{y,\ell^*,b^*}} & \text{if } x \neq x^*, y = y^*, \mathsf{id}_{\ell^*} = b^* \end{cases}$$

Now note that in the second case highlighted above we have that key queries for index $i^* = (x^*, y^*)$ can also be answered as long as the queried identity id satisfies $\mathsf{id}_{\ell^*} \neq b^*$). Since the adversary is constrained not to make key queries of this form as per the lower identity hiding game, thus the reduction algorithm can perfectly simulate lower identity hiding game thereby implying that the scheme satisfies lower identity hiding security assuming that the assumption 1 holds. □

7.1.4 Upper Identity Hiding

In this scenario we need to show that an encryption to (i^*, ℓ^*, b^*) and an encryption to $(i^* + 1, \perp, 0)$ is indistinguishable. Formally, we prove the following lemma.

Lemma 7.4. If the assumptions 1, 2, 3, and 4 holds over the group generator Gen, then no polynomial time adversary can distinguish between an encryption to (i^*, ℓ^*, b^*) and an encryption to $(i^* + 1, \perp, 0)$ with non-negligible advantage (where $i^* = (x^*, y^*)$ is the index output by the adversary in the upper identity hiding game and the adversary is not allowed to make multiple key queries for same index as well as it is not allowed to key query for index i^* with identity id such that $\mathsf{id}_{\ell^*} = 1 - b^*$).

Proof. The proof of this lemma follows by a sequence of hybrid experiments.

Hybrid 1. This hybrid corresponds to the upper identity hiding game in which the challenge ciphertext is an encryption to index-position-value tuple $(i^* = (x^*, y^*), \ell^*, b^*)$.

Hybrid 2. This hybrid is same as previous hybrid, except that the column components in the challenge ciphertext are computed as in Table 18. (Basically in this hybrid the ciphertext component $C_{y^*,\ell^*,1-b^*}$ also includes a random component in the $\mathbb{G}_p$ subgroup. In previous hybrid, only C_{y^*,ℓ^*,b^*} for index j^* included a random component in the $\mathbb{G}_p$ subgroup.)

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > y^*) \vee$ $(y = y^* \wedge \ell \neq \ell^*)$	${H_{y,\ell,b}}^{t} \cdot h^{w_{y,\ell,b}\mathcal{T}}$	$g^{w_{y,\ell,b}}$
$(y < y^*) \vee$ $(y = y^* \wedge \ell = \ell^*)$	${H_{y,\ell,b}}^{t} \cdot h^{w_{y,\ell,b}\mathcal{T}} \cdot {V_{\ell,b}}^{v_{y,\ell,b}\mathcal{T}}$	$g^{w_{y,\ell,b}} \cdot \widetilde{V}_{\ell,b}^{v_{y,\ell,b}}$

Table 18: Computing column components of the ciphertext in Hybrid 2.

Hybrid 3.$(\widetilde{\ell}, \widetilde{b})$ **(for** $\widetilde{\ell} \in [\kappa], \widetilde{b} \in \{0, 1\}$**).** This hybrid is same as previous hybrid, except that the column components in the challenge ciphertext are computed as in Table 19. (Basically in this hybrid the ciphertext components $C_{y^*,\ell,b}$ for $\ell < \widetilde{\ell}$, or $\ell = \widetilde{\ell}$ and $b \leq \widetilde{b}$ also include a random component in the $\mathbb{G}_p$ subgroup.)

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > y^*) \vee$ $(y = y^* \wedge \ell \notin [\widetilde{\ell}] \cup \{\ell^*\}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b > \widetilde{b})$	${H_{y,\ell,b}}^{t} \cdot h^{w_{y,\ell,b}\mathcal{T}}$	$g^{w_{y,\ell,b}}$
$(y < y^*) \vee$ $(y = y^* \wedge \ell \in [\widetilde{\ell} - 1] \cup \{\ell^*\}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b \leq \widetilde{b})$	${H_{y,\ell,b}}^{t} \cdot h^{w_{y,\ell,b}\mathcal{T}} \cdot {V_{\ell,b}}^{v_{y,\ell,b}\mathcal{T}}$	$g^{w_{y,\ell,b}} \cdot \widetilde{V}_{\ell,b}^{v_{y,\ell,b}}$

Table 19: Computing column components of the ciphertext in Hybrid 3.$(\widetilde{\ell}, \widetilde{b})$.

Hybrid 4. (This hybrid is identical to the previous hybrid that is 3.$(\kappa, 1)$. Here we define it explicity only for ease of exposition.) This hybrid is same as the upper identity hiding game, except the challenge ciphertext is an encryption to index-position-value tuple $(i^* = (x^*, y^* + 1), \perp, 0)$.[19]

Hybrid 5. This hybrid corresponds to the upper identity hiding game in which the challenge ciphertext is an encryption to index-position-value tuple $(i^* + 1, \perp, 0)$.[20]

Next, we show via a sequence of claims that the adversary's advantage in each pair of consecutive hybrids is negligibly close thereby completing the proof.

Claim 7.7. If the assumption 1 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 1 and 2 with non-negligible advantage.

Proof. The proof of above claim is identical to that of Lemmas 7.1 and 7.3. □

Claim 7.8. If the assumption 1 holds over the group generator Gen, then for every $\widetilde{\ell} \in [\kappa], \widetilde{b} \in \{0, 1\}$ no polynomial time adversary can distinguish between hybrids 3.$(\widetilde{\ell}, \widetilde{b})$ and 3.$(\widetilde{\ell} + \widetilde{b} - 1, (\widetilde{b} + 1) \bmod 2)$ with non-negligible advantage.

Proof. First, note that if $\widetilde{\ell} = \ell^*$, then we have that hybrids 3.$(\widetilde{\ell}, \widetilde{b})$ and 3.$(\widetilde{\ell} + \widetilde{b} - 1, (\widetilde{b} + 1) \bmod 2)$ are identical. Otherwise, the proof of this lemma can be divided in two cases — (1) either adversary makes a key query of the form (j, id) such that $j = i^*$ and $\mathsf{id}_{\widetilde{\ell}} = \widetilde{b}$, or (2) otherwise. (Note that adversary can make at most one query per index.) Now in the latter case, i.e. case (2), we can use the same proof strategy as used in Lemmas 7.1 and 7.3 which is to use the fact that the reduction algorithm does not need to know the

[19]Note that here y^* could be equal to $\widetilde{m}$. And recall that the special-encryption algorithm can be directly extended to encrypt to such position.

[20]Note that if $y^* \neq \widetilde{m}$, then hybrids 4 and 5 are already identical.

value of group element $g^{r_{x^*} c_{y^*,\widetilde{\ell},\widetilde{b}}}$ for answering the key queries. Whereas in case (1), we provide a different strategy which is to use the fact that to answer key query for index-identity pair (i^*, id), it is sufficient to know the value of group element $g^{r_{x^*}(c_{y^*,\widetilde{\ell},\widetilde{b}} + c_{y^*,\ell^*,b^*})}$ since the adversary is constrained to query for id such that $\mathsf{id}_{\ell^*} = b^*$. Below we formally prove the same.

Type 1: (adversary makes a key query of the form (j, id) such that $j = i^*$ and $\mathsf{id}_{\widetilde{\ell}} = \widetilde{b}$.)
Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$ that distinguishes hybrids $3.(\widetilde{\ell}, \widetilde{b})$ and $3.(\widetilde{\ell} + \widetilde{b} - 1, (\widetilde{b} - 1) \mod 2)$ with non-negligible advantage $\epsilon(\cdot)$. We build a PPT reduction algorithm $\mathcal{B}$ that breaks assumption 1 with advantage $\epsilon(\cdot)$ as follows.

The reduction algorithm $\mathcal{B}$ first receives the modified DBDH challenge from the challenger as

$$(N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), g_q, g_p, A = g_p^a, B = g_p^b, C = g_p^c, D = g_p^{b^2}, T)$$

where T is either g_p^{abc} or a random group element in the subgroup of prime order p. Next, it receives the challenge tuple $(1^\kappa, 1^n, x^*, y^*, \ell^*, b^*)$ from the adversary $\mathcal{A}$.

Now $\mathcal{B}$ needs to generate the public key and send it to the adversary. After that adversary makes polynomially many key queries for distinct indices, and sends a challenge message m. The key queries and challenge message could be arbitrarily interleaved. Below we show that first how does $\mathcal{B}$ generates the public key and later describe how to answer key queries and output challenge ciphertext. Note that finally the adversary outputs its guess, which $\mathcal{B}$ uses to break assumption 1.

Main idea. Since the reduction plays the game with its challenger in the subgroup $\mathbb{G}_p$, thus it can choose everything in the $\mathbb{G}_q$ subgroup by itself. And the main idea is to implicitly set the exponents r_{p,x^*}, s_{p,x^*} as $b \cdot \widetilde{r}_{p,x^*}, \widetilde{s}_{p,x^*}/b$ (respectively) where $\widetilde{r}_{p,x^*}, \widetilde{s}_{p,x^*}$ are chosen at random, and also implicitly set $t_p = a \cdot b$, and $h_p = B = g_p^b$. Most importantly, it sets $c_{p,y^*,\ell^*,b^*} = c + \widetilde{c}_{p,y^*,\ell^*,b^*}$ and $c_{p,y^*,\widetilde{\ell},\widetilde{b}} = -c + \widetilde{c}_{p,y^*,\widetilde{\ell},\widetilde{b}}$, where exponents $\widetilde{c}_{p,y^*,\ell^*,b^*}$, $\widetilde{c}_{p,y^*,\widetilde{\ell},\widetilde{b}}$ are chosen at random. Setting exponents this way allows us to simulate the public key and secret keys exactly as well as the challenge group element T can be programmed in the challenge ciphertext component $C_{y^*,\widetilde{\ell},\widetilde{b}}$.

Public key. It chooses random generator $h_q \in \mathbb{G}_q$ by sampling random exponent $d \in \mathbb{Z}_N$ and setting $h_q = g_q^d$. Additionally it chooses random exponents as follows

$$\begin{aligned} \forall\, \ell \in [\kappa], b \in \{0,1\}, &\qquad \delta_{\ell,b} \leftarrow \mathbb{Z}_N, \quad \gamma_{\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, &\qquad \widetilde{c}_{y,\ell,b} \leftarrow \mathbb{Z}_N \\ \forall\, x \in [m], &\qquad \widetilde{r}_x \leftarrow \mathbb{Z}_N,\ \alpha_x \leftarrow \mathbb{Z}_N \end{aligned}$$

It also samples $\beta \leftarrow \mathbb{Z}_N$. Next it computes key components $\{E_x, F_x, G_x, E_{q,x}, F_{q,x}, G_{q,x}\}$, $\{H_{y,\ell,b}\}$ for $x \in [m]$ and $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$ as

$$E_x = \begin{cases} (g_p g_q)^{\widetilde{r}_x} & \text{for } x \neq x^*, \\ (B g_q)^{\widetilde{r}_x} & \text{otherwise.} \end{cases}, \qquad F_x = \begin{cases} (B h_q)^{\widetilde{r}_x} & \text{for } x \neq x^*, \\ (D h_q)^{\widetilde{r}_x} & \text{otherwise.} \end{cases},$$

$$H_{y,\ell,b} = \begin{cases} C (g_p g_q)^{\widetilde{c}_{y,\ell,b}} & \text{for } y = y^*, \ell = \ell^*, b = b^*, \\ C^{-1} (g_p g_q)^{\widetilde{c}_{y,\ell,b}} & \text{for } y = y^*, \ell = \widetilde{\ell}, b = \widetilde{b}, \\ (g_p g_q)^{\widetilde{c}_{y,\ell,b}} & \text{otherwise.} \end{cases},$$

$$E_{q,x} = g_q^{\beta \widetilde{r}_x},\ F_{q,x} = h_q^{\beta \widetilde{r}_x},\ G_{q,x} = e(g_q, g_q)^{\beta \alpha_x},\ G_x = e(g,g)^{\alpha_x}.$$

Finally, it sets the public key as

$$\mathsf{pk} = \left(\begin{array}{c} N, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot), \quad g = g_p g_q, h = B h_q, E_q = g_q^{\beta}, \\ \left\{ \begin{array}{c} E_x,\ F_x,\ G_x, \\ E_{q,x},\ F_{q,x},\ G_{q,x} \end{array} \right\}_x, \{H_{y,\ell,b}\}_{y,\ell,b}, \\ \left\{ \widetilde{V}_{\ell,b} = g^{\delta_{\ell,b}} g_p^{\gamma_{\ell,b}},\ V_{\ell,b} = h^{\delta_{\ell,b}} \right\}_{\ell,b} \end{array} \right)$$

Note that all the above terms can be computed using only the modified DBDH challenge.

Secret key queries. $\mathcal{B}$ answers the key query for index-identity pair $(i = (x,y), \mathsf{id})$ with $\mathsf{sk} = (x, y, \mathsf{id}, K_{x,y})$ where $K_{x,y}$ is computed as follows

$$K_{x,y} = \begin{cases} g^{\alpha_x} g^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x \neq x^*, y \neq y^* \\ g^{\alpha_x} (B g_q)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x = x^*, y \neq y^* \\ g^{\alpha_x} (\prod_\ell H_{y,\ell,\mathsf{id}_\ell})^{\widetilde{r}_x} & \text{if } x \neq x^*, y = y^* \\ g^{\alpha_x} (B g_q)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x = x^*, y = y^*, \mathsf{id}_{\ell^*} \neq b^*, \text{ and } \mathsf{id}_{\widetilde{\ell}} \neq \widetilde{b} \\ g^{\alpha_x} (B g_q)^{\widetilde{r}_x (\sum_{\ell \in [\kappa]} \widetilde{c}_{y,\ell,\mathsf{id}_\ell})} & \text{if } x = x^*, y = y^*, \mathsf{id}_{\ell^*} = b^*, \text{ and } \mathsf{id}_{\widetilde{\ell}} = \widetilde{b} \end{cases}$$

Note that the adversary is not allowed to query for index-identity pair (j, id) such that $j = i^* = (x^*, y^*)$ and $\mathsf{id}_{\ell^*} \neq b^*$. Also, note that Type 1 adversaries make a key queries of the form (j, id) such that $j = i^*$ and $\mathsf{id}_{\widetilde{\ell}} = \widetilde{b}$. That is, if the adversary queries for (i^*, id) such that $\mathsf{id}_{\widetilde{\ell}} \neq \widetilde{b}$, then $\mathcal{B}$ aborts.

Challenge ciphertext. It chooses random exponents as follows

$$\begin{array}{lll} \tau \in \mathbb{Z}_N,\ t_q \leftarrow \mathbb{Z}_N, & \forall\, x \in [m], & e_x, f_x \leftarrow \mathbb{Z}_N, \widetilde{s}_x \leftarrow \mathbb{Z}_N \\ & \forall\, y \in [\widetilde{m}], \ell \in [\kappa], b \in \{0,1\}, & \widetilde{w}_{y,\ell,b} \leftarrow \mathbb{Z}_N, v_{y,\ell,b} \leftarrow \mathbb{Z}_N \end{array}$$

Now the ciphertext components are computed as described in Tables 20 and 21.

	R_x	$\widetilde{R}_x$	A_x	B_x
$> x^*$	${E_{q,x}}^{\widetilde{s}_x}$	${F_{q,x}}^{\widetilde{s}_x \tau}$	${E_q}^{\widetilde{s}_x t_q}$	$M \cdot {G_{q,x}}^{\widetilde{s}_x t_q}$
$= x^*$	$g^{\widetilde{r}_x \widetilde{s}_x}$	$(B h_q)^{\widetilde{r}_x \widetilde{s}_x \tau}$	$(A g_q^{t_q})^{\widetilde{s}_x}$	$M \cdot e(g, A g_q^{t_q})^{\alpha_x \widetilde{s}_x}$
$< x^*$	$g^{\widetilde{s}_x}$	$(B h_q)^{\widetilde{s}_x \tau}$	g^{e_x}	$e(g,g)^{f_x}$

Table 20: Computing row components of the ciphertext for $x \in [m]$.

	$C_{y,\ell,b}$	$\widetilde{C}_{y,\ell,b}$
$(y > y^*) \vee$ $(y = y^* \wedge \ell \notin [\widetilde{\ell}] \cup \{\ell^*\}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b > \widetilde{b})$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau}$	$A^{-\widetilde{c}_{y,\ell,b}/\tau} g^{\widetilde{w}_{y,\ell,b}}$
$y = y^* \wedge \ell = \widetilde{\ell} \wedge b = \widetilde{b}$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau} T^{-1}$	$A^{-\widetilde{c}_{y,\ell,b}/\tau} g^{\widetilde{w}_{y,\ell,b}}$
$(y < y^*) \vee$ $(y = y^* \wedge \ell \in [\widetilde{\ell} - 1] \cup \{\ell^*\}) \vee$ $(y = y^* \wedge \ell = \widetilde{\ell} \wedge b < \widetilde{b})$	$g_q^{\widetilde{c}_{y,\ell,b} t_q} h^{\widetilde{w}_{y,\ell,b} \tau} g_p^{v_{y,\ell,b}}$	$g^{\widetilde{w}_{y,\ell,b}}$

Table 21: Computing column components of the ciphertext for $(y, \ell, b) \in [\widetilde{m}] \times [\kappa] \times \{0,1\}$.

Finally, $\mathcal{B}$ receives a guess b' from $\mathcal{A}$ and it simply forwards that as its guess to the modified DBDH challenger.

Analysis. If $T = g_p^{abc}$, then $\mathcal{B}$ simulates the view of hybrid $H_{\widetilde{\ell}+\widetilde{b}-1,(\widetilde{b}-1) \bmod 2}$, otherwise if T is randomly chosen group element in $\mathbb{G}_p$ subgroup then $\mathcal{B}$ simulates the view of hybrid $H_{\widetilde{\ell},\widetilde{b}}$. Therefore, if $\mathcal{A}$ wins with advantage ϵ, then $\mathcal{B}$ breaks assumption 1 with advantage ϵ.

Type 2: (otherwise.) The proof is identical to that of Lemmas 7.1 and 7.3.

□

Claim 7.9. No adversary can distinguish between hybrids 3 and 4 with non-negligible advantage.

Proof. The proof of this claim is immediate as hybrids 3 and 4 are identical. □

Claim 7.10. If the assumptions 1, 2, 3, and 4 holds over the group generator Gen, then no polynomial time adversary can distinguish between hybrids 4 and 5 with non-negligible advantage.

Proof. First, note that if $y^* \neq \widetilde{m}$, then hybrids 4 and 5 are already identical. Otherwise, the proof of this claim follows by a sequence of hybrid experiments and is identical to the proof of Lemma 7.2. Concretely, hybrid 4 described above is identical to the *hybrid 2* described in the proof of Lemma 7.2. Similarly, hybrid 5 described above is identical to the *hybrid 7* described in the proof of Lemma 7.2. Thus, the proof of this claim follows directly from the proof of indistinguishability of *hybrids* 2 *and* 7 from Lemma 7.2. □

This concludes the proof of Lemma 7.4. □

7.1.5 Message Hiding

First, observe that the special-encryption ciphertexts for index-position-value tuple $(m \cdot \widetilde{m} + 1, \perp, 0)$ are independent of the message M encrypted. Now suppose $m \cdot \widetilde{m} > n$, then using the strategy same as used in the proof of index hiding security we can indistinguishably switch ciphertexts from encryptions for index-position-value tuple $(n+1, \perp, 0)$ to $(m \cdot \widetilde{m} + 1, \perp, 0)$. This is because each of the adversary's key query must be of the form (i, id) where $i \in [n]$. Thus, the message hiding security follows by a simple hybrid argument.

8 Building EIPLBE from the Learning with Errors Assumption

We will now discuss how to build EIPLBE with ciphertext size polylogarithmic in the index bound n. For this construction, we require the notion of mixed functional encryption, introduced by [GKW18]. Here, we use the syntax and definitions from [CVW$^+$18].

8.1 Secret-key and Mixed Functional Encryption

***t*-CT SKFE.** We begin with the definition for SKFE:

Definition 8.1 (Secret-key functional encryption (SKFE))**.** A secret-key functional encryption scheme for a class of functions $\mathcal{F}_\mu = \{f : \{0,1\}^\mu \to \{0,1\}\}$ is a tuple of probabilistic polynomial time (p.p.t) algorithms $(\mathsf{Setup}, \mathsf{SKGen}, \mathsf{SKEnc}, \mathsf{Dec})$ such that:

- $\mathsf{Setup}(1^\lambda)$ takes as input the security parameter 1^λ, and outputs the master secret key MSK and the public parameters PP.
- $\mathsf{SKGen}(\mathsf{MSK}, m)$ takes as input MSK and a message $m \in \{0,1\}^\mu$, and outputs a decryption key SK_m.
- $\mathsf{SKEnc}(\mathsf{MSK}, f)$ takes as input MSK and a function $f \in \mathcal{F}_\mu$, and outputs a ciphertext CT.
- $\mathsf{Dec}(\mathsf{SK}_m, \mathsf{CT})$ takes as input SK_m and CT, and outputs a single bit.

Correctness. For every message $m \in \{0,1\}^\mu$ and function $f \in \mathcal{F}_\mu$ we have:

$$\begin{aligned}\Pr[&\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda);\ \mathsf{SK}_m \leftarrow \mathsf{SKGen}(\mathsf{MSK}, m) :\\ &\mathsf{Dec}(\mathsf{SK}_m, \mathsf{SKEnc}(\mathsf{MSK}, f)) = f(m)] = 1 - \mathsf{negl}(\lambda),\end{aligned}$$

where the probability is taken over the randomness of the algorithms $\mathsf{Setup}, \mathsf{SKGen}, \mathsf{SKEnc}, \mathsf{Dec}$.

Function-hiding security. For all p.p.t stateful algorithms Adv, there is a p.p.t. stateful algorithm Sim such that:

$$\left\{\text{Experiment REAL}_{\mathsf{Adv}}(1^\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\text{Experiment IDEAL}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)\right\}_{\lambda\in\mathbb{N}}$$

where the real and ideal experiments of stateful algorithms $\mathsf{Adv}, \mathsf{Sim}$ are as follows:

Experiment $\text{REAL}_{\mathsf{Adv}}(1^\lambda)$	Experiment $\text{IDEAL}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$
$\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda)$,	$\mathsf{Sim} \leftarrow 1^\lambda$
For $i \in [t]$:	For $i \in [t]$:
$\mathsf{Adv} \rightarrow f^{[i]}$;	$\mathsf{Adv} \rightarrow f^{[i]}$;
$\mathsf{Adv} \leftarrow \mathsf{CT}^{[i]} = \mathsf{SKEnc}(\mathsf{PP}, \mathsf{MSK}, f^{[i]})$;	$\mathsf{Adv} \leftarrow \mathsf{CT}^{[i]} = \mathsf{Sim}(1^{\lvert f^{[i]}\rvert})$;
Repeat polynomially many times:	Repeat polynomially many times:
$\mathsf{Adv} \rightarrow m;\ \mathsf{Adv} \leftarrow \mathsf{SKGen}(\mathsf{PP}, \mathsf{MSK}, m)$	$\mathsf{Adv} \rightarrow m;\ \mathsf{Adv} \leftarrow \mathsf{Sim}(m, \left\{f^{[i]}(m)\right\}_{i\in[t]})$
$\mathsf{Adv} \rightarrow b$; Output b	$\mathsf{Adv} \rightarrow b$; Output b

In the experiments, the adversary Adv can ask for t ciphertexts followed by polynomially many decryption key queries. Once Adv makes a ciphertext query for a function $f \in \mathcal{F}_\lambda$, in the real experiment Adv obtains the ciphertext generated by the secret-key encryption algorithm; in the ideal experiment Adv obtains the ciphertext generated by Sim given only the (circuit) size of f. Once Adv makes a message query m, in the real experiment Adv obtains SK_m from the decryption key generation algorithm; in the ideal experiment, Adv obtains the decryption key generated by the simulator who is given m, and $\left\{f^{[i]}(m)\right\}_{i\in[t]}$. The output of the experiment is the final output bit of Adv.

Remark 8.1 (adaptive security)**.** A t-CT SKFE scheme is called adaptively secure if the function and ciphertext queries can be made adaptively in any order.

***t*-CT mixed FE.** We provide a simulation-based definition for t-ciphertext (t-CT) mixed-FE, which is same as the definition in [GKW18, Section 5] where it is referred to as $(t-1)$-bounded mixed-FE.

Definition 8.2 (Mixed functional encryption)**.** A mixed functional encryption scheme for a class of functions $\mathcal{F}_\mu = \{f : \{0,1\}^\mu \rightarrow \{0,1\}\}$ is a tuple of probabilistic polynomial time (p.p.t) algorithms (Setup, SKGen, SKEnc, Dec, PKEnc) such that:

- $(\mathsf{Setup}, \mathsf{SKGen}, \mathsf{SKEnc}, \mathsf{Dec})$ are the same as SKFE.
- $\mathsf{PKEnc}(\mathsf{PP})$ takes as input PP, and outputs a ciphertext CT.

Correctness and Function-hiding security. Same as SKFE.

Public/secret-key mode indistinguishability. In addition to the security requirement above for a normal secret-key functional encryption, a mixed-FE further requires that for a function f queried to the encryption oracle, if for all message m queried by the adversary, $f(m) = 1$ (the other potential $t-1$ functions does not have to satisfy this requirement), then the secret-key ciphertext $\mathsf{SKEnc}(\mathsf{MSK}, f)$ is indistinguishable from a sample from $\mathsf{PKEnc}(\mathsf{PP})$. Formally, we require that for all p.p.t stateful algorithms Adv, the following two experiments produce indistinguishable outputs:

$$\left\{\text{Experiment SKEXP}_{\mathsf{Adv}}(1^\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\text{Experiment PKEXP}_{\mathsf{Adv}}(1^\lambda)\right\}_{\lambda\in\mathbb{N}}$$

The experiments are as follows:

```
Experiment SKEXP_Adv(1^λ)
MSK ← Gen(1^λ),
For i in [i* − 1]:
Adv → f^[i];
  Adv ← CT^[i] = SKEnc(MSK, f^[i]);
Adv → f^[i*];
  Adv ← CT^[i*] = SKEnc(MSK, f^[i*]);
For i in [i* + 1, t]:
Adv → f^[i];
  Adv ← CT^[i] = SKEnc(MSK, f^[i]);
Repeat polynomially many times:
   Adv → m; Adv ← SKGen(MSK, m)
Adv → b; Output b
```

```
Experiment PKEXP_Adv(1^λ)
MSK ← Gen(1^λ),
For i in [i* − 1]:
Adv → f^[i];
  Adv ← CT^[i] = SKEnc(MSK, f^[i]);
Adv → f^[i*];
  Adv ← CT^[i*] = PKEnc(PP);
For i in [i* + 1, t]:
Adv → f^[i];
  Adv ← CT^[i] = SKEnc(MSK, f^[i]);
Repeat polynomially many times:
   Adv → m; Adv ← SKGen(MSK, m)
Adv → b; Output b
```

8.2 Building EIPLBE from ABE and Mixed FE

Goyal et al. [GKW18] showed how to build PLBE from ABE and mixed functional encryption. They used mixed FE for the comparison function class. The same transformation can also be used to achieve EIPLBE if we use the following (more expressive) function family:

$$\left\{f_{y^*,\ell,b}(y,\mathsf{id}) = 1 \text{ iff } y > y^* \text{ or } (y,\mathsf{id}_\ell) = (y^*,\bar{b})\right\}_{y^*,\ell,b}$$

This, combined with the LWE based ABE schemes [GVW13, BGG+14] and the LWE based Mixed FE schemes [GKW18, CVW+18], results in an LWE based EIPLBE scheme with succinct ciphertexts.

9 Bounded Embedded-Identity Traitor Tracing

We will now present the syntax and definitions for traitor tracing with embedded identities where the key generation is *not* indexed, but the number of key queries is bounded.

Let $\mathcal{T}$ be a (bounded keys, public/private tracing)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$. It consists of five algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}}) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$: The setup algorithm takes as input the security parameter λ, identity space index κ, bound on number of key queries n_{bd}, and outputs a master secret key msk and a public key pk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa) \to \mathsf{sk}_{\mathsf{id}}$: The key generation algorithm takes as input the master secret key and identity $\mathsf{id} \in \{0,1\}^\kappa$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$: The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$: The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1) \to T \subseteq \{0,1\}^\kappa$. The trace algorithm has oracle access to a program D, it takes as input key (which is the master secret key in a private tracing scheme, and the public key in a public tracing scheme), parameter y and two messages m_0, m_1. It outputs a set T of identities, where $T \subseteq \{0,1\}^\kappa$.

Correctness. A traitor tracing scheme is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, \kappa, n_{\mathrm{bd}} \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$ and identity $\mathsf{id} \in \{0,1\}^\kappa$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}}); \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Efficiency Let $\mathsf{T\text{-}s}, \mathsf{T\text{-}e}, \mathsf{T\text{-}k}$ be functions. A (bounded keys, public/private tracing)-embedded identity tracing scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$)-efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}})$ is at most $\mathsf{T\text{-}s}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The number of oracle calls made by $\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1)$ to decoding box D is at most $\mathsf{T\text{-}t}(\lambda, \kappa, n_{\mathrm{bd}}, y)$.
- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa, n_{\mathrm{bd}})$.

9.1 Security

The $\mathsf{IND\text{-}CPA}$ security definition is identical to the one in Section 4.1.1; the tracing-based security definition is very similar, but instead of requiring that the index queries are distinct, we require that the number of queries in the 'correct-trace experiment' is at most n_{bd}.

Definition 9.1 (Secure tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (bounded keys, public/private tracing)-embedded identity tracing scheme. For any non-negligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}bd}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Figure 4.

Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}bd}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^\kappa, 1^{n_{\mathrm{bd}}} \leftarrow \mathcal{A}(1^\lambda)$.
- $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}})$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$.
- $T \leftarrow \mathsf{Trace}^D(\mathsf{key}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$.

Let $S_{\mathcal{ID}}$ be the set of identities queried by $\mathcal{A}$. Here, $O(\cdot)$ is an oracle that has msk hardwired, takes as input an identity $\mathsf{id} \in \{0,1\}^\kappa$ and outputs $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$.

Figure 4: Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}bd}$

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$
  $\mathsf{Admissible\text{-}Adv}$: $\mathcal{A}$ makes at most n_{bd} key queries
  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}]$.

- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathcal{ID}}$
  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.
- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S_{\mathcal{ID}}$
  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A traitor tracing scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathsf{negl}_1(\cdot)$, $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathsf{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}_2(\lambda).$$

9.2 Going from Indexed to Bounded

In this section, we will show how to build a (bounded keys, public/private tracing)-embedded identity tracing scheme from a (indexed keygen, public/private)-embedded identity tracing scheme.

9.2.1 Construction

Let $\mathsf{TT\text{-}ind} = (\mathsf{Ind.Setup}, \mathsf{Ind.Enc}, \mathsf{Ind.KeyGen}, \mathsf{Ind.Dec}, \mathsf{Ind.Trace})$ be a (indexed keygen, public/private)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$, and with (T-s, T-e, T-k, T-d, T-t, S-c, S-k)-efficiency, and let $\mathcal{S} = (\mathsf{SS.Setup}, \mathsf{SS.Sign}, \mathsf{SS.Verify})$ be a signature scheme with message space $\{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$ and signature space $\left\{\{0,1\}^{\ell_{\mathsf{ss}}(\lambda)}\right\}_{\lambda\in\mathbb{N}}$. We use $\mathsf{TT\text{-}ind}$ to build a non-indexed traitor tracing scheme $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ as follows.

$\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}}) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the $\mathsf{TT\text{-}ind}$ setup algorithm λ times with index value $n_{\mathrm{indx}} = 2 \cdot n_{\mathrm{bd}}^2$ and identity index $\kappa + \ell_{\mathsf{ss}}(\lambda)$ as follows:

$$\forall i \in [\lambda], \quad (\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{Ind.Setup}(1^\lambda, 1^{\kappa+\ell_{\mathsf{ss}}(\lambda)}, n_{\mathrm{indx}} = 2 \cdot n_{\mathrm{bd}}^2).$$

It also chooses a signing/verification keys $(\mathsf{ss.sk}, \mathsf{ss.vk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^\kappa)$. It then sets the master secret and public keys as $\mathsf{msk} = \left((\mathsf{msk}_i)_{i\in[\lambda]}, \mathsf{ss.sk}, \mathsf{ss.vk}\right)$, $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$ and $\mathsf{key} = \mathsf{ss.vk}, (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}$. Let $\mathsf{msk} = \left((\mathsf{msk}_i)_{i\in[\lambda]}, \mathsf{ss.sk}, \mathsf{ss.vk}\right)$. The key generation algorithm chooses λ uniformly random indices $j_i \leftarrow \left[2 \cdot n_{\mathrm{bd}}^2\right]$ for $i \in [\lambda]$. Next, it computes a signature on id; that is, it computes $\sigma \leftarrow \mathsf{SS.Sign}(\mathsf{ss.sk}, \mathsf{id})$. Finally, for i^{th} subsystem, it computes indexed keys for index $\{j_i\}$ and identity (id, σ). That is, for $i \in [\lambda]$, it computes $\mathsf{sk}_i \leftarrow \mathsf{Ind.KeyGen}(\mathsf{msk}_i, (\mathsf{id}, \sigma), j_i)$, and outputs the secret key sk as $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\lambda]}$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. Let $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$. The encryption algorithm first chooses $\lambda - 1$ random messages as $r_i \leftarrow \mathcal{M}$ for $i \in [\lambda - 1]$. Next, it sets $r_\lambda = m \oplus \left(\bigoplus_{i=1}^{\lambda-1} r_i\right)$. It then encrypts messages r_i under key pk_i as follows:

$$\forall i \in [\lambda], \quad \mathsf{ct}_i \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_i, r_i).$$

Finally, it outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. Let $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\lambda]}$, and $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$. The decryption algorithm runs the $\mathsf{TT\text{-}ind}$ decryption on each secret key-ciphertext pair as $z_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_i)$ for $i \in [\lambda]$.

If $z_i = \perp$ for any $i \in [\lambda]$, then it outputs $z = \perp$, otherwise it outputs $z = \bigoplus_{i=1}^{\lambda} z_i$ as the message.

$\mathsf{isGoodDecoder}^{D}\left((\mathsf{pk}_i)_{i\in[\lambda]}, 1^y, m_0, m_1, r, i\right)$

Input: Keys $(\mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: Yes/No.

```
1. Set $\mathsf{count} = 0$. (Let $\epsilon = 1/y$.)
2. For $j = 1$ to $\lambda \cdot y$:
    - Choose $\lambda - 1$ messages $r_k$ randomly for $k \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{k\in[\lambda]\setminus\{i\}} r_k = r$. (That is, bit-wise parity of the messages chosen matches the message $r$.)
    - Choose random bit $b \leftarrow \{0,1\}$, and compute ciphertexts as $\mathsf{ct}_k \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_k, r_k)$ for $k \in [\lambda]\setminus\{i\}$, and $\mathsf{ct}_i \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_i, r \oplus m_b)$.
    - Query ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda)$ to the oracle $D$. Let $b'$ denote the oracle's response.
    - If $b = b'$, set $\mathsf{count} = \mathsf{count} + 1$.
3. If $\mathsf{count}/(\lambda \cdot y) \geq 1/2 + \epsilon/3$, then output 'Yes'. Otherwise output 'No'.
```

Figure 5: Routine $\mathsf{isGoodDecoder}$

$\mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$

Input: Keys $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: $T \subseteq \{0,1\}^\kappa$.

1. It runs the $\mathsf{TT\text{-}ind}$ tracing algorithm on inputs — keys $\mathsf{key}_i, \mathsf{pk}_i$, parameter $4y$, messages $m_0 \oplus r, m_1 \oplus r$ — and with oracle access to oracle $\widetilde{D}$ which we define next.
2. On each query ct to oracle $\widetilde{D}$ by $\mathsf{Ind.Trace}$, the sub-tracing algorithm first chooses $\lambda - 1$ messages r_j randomly for $j \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r$. (That is, bit-wise parity of the messages chosen matches the message r.) It encrypts these messages as $\mathsf{ct}_j \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_j, r_j)$, and then sends the ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda)$ to the oracle D as its query. And it forwards the oracle D's response to the $\mathsf{Ind.Trace}$ algorithm.
3. Finally, the $\mathsf{Ind.Trace}$ algorithm outputs a set T. The sub-tracing algorithm outputs the same set T as its output.

In short, $\mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i) = \mathsf{Ind.Trace}^{\widetilde{D}}((\mathsf{msk}_i, \mathsf{pk}_i), 1^{4y}, m_0 \oplus r, m_1 \oplus r),$

$$\widetilde{D}(\mathsf{ct}) = D(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda),$$
$$\text{where} \quad \forall j \in [\lambda] \setminus \{i\},\ r_j \leftarrow \mathcal{M} \text{ such that } \bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r,$$
$$\forall j \in [\lambda] \setminus \{i\},\ \mathsf{ct}_j \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_j, r_j)$$

Figure 6: Routine $\mathsf{SubTrace}$

$\mathsf{Trace}^{D}(\mathsf{key}, 1^y, m_0, m_1) \to T$. Let $\mathsf{key} = \left((\mathsf{msk}_i, \mathsf{pk}_i)_{i\in[\lambda]}, \mathsf{ss.vk}\right)$ and $\epsilon = 1/y$. First we define a supplementary algorithms $\mathsf{isGoodDecoder}$ (in Figure 5) and $\mathsf{SubTrace}$ (in Figure 6) that both get oracle access to the decoder D and take as input all λ public-secret key pairs $(\mathsf{msk}_i, \mathsf{pk}_i)_i$, parameter y, messages m_0, m_1, r and an index $i \in [\lambda]$. The tracing algorithm executes the following procedure using $\mathsf{isGoodDecoder}$ and $\mathsf{SubTrace}$ routines as follows:

```
1. Set $i = 1$.
2. Set $\mathsf{flag}$ = 'No'. For $j = 1$ to $\lambda \cdot y$:
    - Choose a random message $r \leftarrow \mathcal{M}$.
    - Run $\mathsf{isGoodDecoder}$ as $\mathsf{flag} \leftarrow \mathsf{isGoodDecoder}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$.
    - If $\mathsf{flag}$ = 'Yes', break. Else, continue.
3. If $\mathsf{flag}$ = 'Yes', run $\mathsf{SubTrace}$ as $T \leftarrow \mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$. Else, set $T = \emptyset$.
```

4. If $T = \emptyset$ and $i < \lambda$, set $i = i + 1$ and go to step 2. Otherwise, output T.

Set $T^{\text{final}} = \emptyset$. For each $(\mathsf{id}, \sigma) = \tilde{\mathsf{id}} \in T$, if $\mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1$, the tracing algorithm adds id to T^{final}. Concretely,

$$T^{\text{final}} = \{\mathsf{id} \ : \ \exists \, \sigma \text{ such that } (\mathsf{id}, \sigma) \in T \text{ and } \mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1\}\,.$$

Finally, it outputs the set T^{final} as the set of traitors.

Next, we prove the following.

Theorem 9.1. If $\mathsf{TT\text{-}ind} = (\mathsf{Ind.Setup}, \mathsf{Ind.Enc}, \mathsf{Ind.KeyGen}, \mathsf{Ind.Dec}, \mathsf{Ind.Trace})$ is a secure (indexed keygen, public/private)-embedded identity tracing scheme (as per Definitions 4.2 and B.2) with ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)-efficiency, and $\mathcal{S} = (\mathsf{SS.Setup}, \mathsf{SS.Sign}, \mathsf{SS.Verify})$ is a secure signature scheme (as per Definition A.1), then the scheme $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ (described in Section 9.2.1) is a secure (bounded keys, public/private tracing)-embedded identity tracing scheme (as per Definitions 4.2 and 9.1) with ($\mathsf{T\text{-}s}'$, $\mathsf{T\text{-}e}'$, $\mathsf{T\text{-}k}'$, $\mathsf{T\text{-}d}'$, $\mathsf{T\text{-}t}'$, $\mathsf{S\text{-}c}'$, $\mathsf{S\text{-}k}'$)-efficiency, where the efficiency measures are related as follows:

- $\mathsf{T\text{-}s}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}s}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}k}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}k}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}e}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}e}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}d}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}d}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}t}'(\lambda, \kappa, n, y) = \lambda \cdot \mathsf{T\text{-}t}(\lambda, \kappa, 2 \cdot n^2, 4y) + \lambda^3 \cdot y^2$,
- $\mathsf{S\text{-}c}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{S\text{-}c}(\lambda, \kappa, 2 \cdot n^2)$,
- $\mathsf{S\text{-}k}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{S\text{-}k}(\lambda, \kappa, 2 \cdot n^2)$.

Proof. **Correctness**: Fix any security parameter λ, public key $\mathsf{pk} = (\mathsf{pk}_i)_{i \in [\lambda]}$, master secret key $\mathsf{msk} = \Big((\mathsf{msk}_i)_{i \in [\lambda]}, (\mathsf{ss.sk}, \mathsf{ss.vk})\Big)$, tracing key $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)$, message $m \in \mathcal{M}$ and identity id. The encryption algorithm chooses $\{r_i\}_{i \in [\lambda]}$ such that $\bigoplus_i r_i = m$, computes $\mathsf{ct}_i \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_i, r_i)$ and sets $\mathsf{ct} = (\mathsf{ct}_i)_{i \in [\lambda]}$. The key generation algorithm first chooses λ indices $\{j_i\}_{i \in [\lambda]}$, computes a signature σ on id, and outputs λ keys $(\mathsf{sk}_i)_{i \in [\lambda]}$, where sk_i is a key for identity (id, σ) and index j_i. From the correctness of the underlying scheme $\mathsf{TT\text{-}ind}$, it follows that decryption of ct_i using sk_i outputs r_i. As a result, the decryption of ct using sk outputs message m.

IND-CPA security: $\mathsf{IND\text{-}CPA}$ security of our scheme follows directly from the $\mathsf{IND\text{-}CPA}$ security of $\mathsf{TT\text{-}ind}$. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks the $\mathsf{IND\text{-}CPA}$ security of our scheme with advantage ϵ. We can use $\mathcal{A}$ to break the $\mathsf{IND\text{-}CPA}$ security of $\mathsf{TT\text{-}ind}$ with advantage ϵ. The reduction algorithm first receives pk_1 from the $\mathsf{TT\text{-}ind}$ challenger. It chooses $(\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{Ind.Setup}(1^\lambda, 1^\kappa, n_{\text{indx}})$ for each $i \in \{2, \ldots, \lambda\}$, and sends $(\mathsf{pk}_i)_{i \in [\lambda]}$ to $\mathcal{A}$. (If $\mathsf{TT\text{-}ind}$ is a public tracing scheme, then the reduction algorithm also receives a tracing key key_1 from the challenger, and it sends the tracing key $(\mathsf{key}_i, \mathsf{pk}_i)_{i \in [\lambda]}$ to $\mathcal{A}$).

Next, $\mathcal{A}$ sends two messages m_0, m_1 to $\mathcal{B}$. The reduction algorithm chooses $r_2, \ldots, r_n \leftarrow \mathcal{M}$, sets $m'_b = m_b \oplus \left(\bigoplus_{i>1} r_i\right)$ and sends (m'_0, m'_1) to the challenger. The challenger sends ct_1 to $\mathcal{B}$. The reduction algorithm computes encryptions of $r_2, \ldots, r_\lambda$, and sends $\mathsf{ct} = (\mathsf{ct}_i)_{i \in [\lambda]}$ to $\mathcal{A}$. The adversary sends its guess, which the reduction algorithm forwards to the challenger.

Clearly, if $\mathcal{A}$ has advantage ϵ, then so does $\mathcal{B}$.

Efficiency: It is easy to verify that $\mathsf{T\text{-}s}'$, $\mathsf{T\text{-}k}'$, $\mathsf{T\text{-}e}'$, $\mathsf{T\text{-}d}'$, $\mathsf{T\text{-}t}'$, $\mathsf{S\text{-}c}'$ and $\mathsf{S\text{-}k}'$ satisfy the efficiency measures.

Correct Trace and False Trace guarantees: We will now show that our scheme satisfies Definition 9.1.

False Trace: We will first show that the false trace probability of our scheme is negligible in the security parameter. Recall, the false trace probability is defined as follows: the adversary receives the public key (and the tracing key in a public tracing algorithm). Next, it queries for a set of keys $S_{\mathcal{ID}}$ corresponding to a set of identities (note that in the false trace experiment, the adversary can query for an unbounded number of keys). Finally, it outputs a decoding box D and two messages m_0, m_1. The tracing algorithm uses D and m_0, m_1 to output a set of traitors T^{final}. In particular, the tracing algorithm uses the $\mathsf{SubTrace}$ algorithm to find a polynomial sized set $T = \{(\mathsf{id}_j, \sigma_j)\}_j$, and puts an identity id in T^{final} if $(\mathsf{id}, \sigma) \in T$ and $\mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1$. The false trace probability is the probability that $T^{\text{final}} \not\subseteq S_{\mathcal{ID}}$.

Suppose there exists a PPT adversary $\mathcal{A}$ such that the false trace probability is ϵ. We can use $\mathcal{A}$ to break the signature scheme's security. The reduction algorithm $\mathcal{B}$ receives the verification key $\mathsf{ss.vk}$ from the challenger. It chooses λ public/master secret/tracing keys $\{(\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i)\}$ for the underlying traitor tracing system $\mathsf{TT\text{-}ind}$, and sends $(\mathsf{pk}_i)_{i\in[\lambda]}$ to the adversary (if $\mathsf{TT\text{-}ind}$ is public tracing, then it also sends $\left((\mathsf{key}_i)_{i\in[\lambda]}, \mathsf{ss.vk}\right)$ as the tracing key). Next, the adversary gets access to a key generation oracle. For every query id, the reduction algorithm sends id to the challenger, and receives signature σ. It then chooses λ uniformly random indices $\{j_i\}_{i\in[\lambda]}$ and generates indexed keys $\mathsf{sk}_j \leftarrow \mathsf{Ind.KeyGen}(\mathsf{msk}_i, (\mathsf{id}, \sigma), j_i)$ for each $i \in [\lambda]$. It sends $(\mathsf{sk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. Finally, after polynomially many secret key queries, the adversary outputs a decoding box D and messages m_0, m_1. The reduction algorithm first checks if it is a good decoder (that is, steps 1 and 2 of the tracing algorithm). Next, if it is a good decoder, it runs $\mathsf{SubTrace}$ to obtain a set $T = \{\mathsf{id}_j, \sigma_j\}_j$. If there exists an (id, σ) pair in T such that id was not queried by $\mathcal{A}$ and $\mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1$, then it sends id to the challenger. Since the false trace event happens with probability ϵ, the reduction algorithm breaks the security of the signature scheme with probability ϵ.

Remark 9.1. Note that we do not require the 'false trace' guarantee of the underlying tracing scheme $\mathsf{TT\text{-}ind}$. We only require that the tracing algorithm of the underlying scheme is polynomial time.

Correct Trace:

Recall the traitor tracing experiment (Figure 4): the challenger sends a public key pk, then the adversary queries for a set of secret keys (at most n_{bd} keys) corresponding to identities. Finally, the adversary outputs a decoding box D. We need to show that if the decoding box if ϵ good, then the tracing algorithm can trace at least one identity for which the adversary queried a secret key. More formally, the correct trace guarantee is captured by the following events/probabilities:

$\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \;:\; b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$

$\mathsf{Admissible\text{-}Adv}$: $\mathcal{A}$ makes at most n_{bd} key queries

$\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}]$.

$\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathcal{ID}}$, $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

We require that $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}_2(\lambda)$.

We will define some more events and their probabilities, which will be useful for our proof.

- Tracing without correctness: Event Tr, which is similar to $\mathsf{Cor\text{-}Tr}$, except that we do not require $T \subseteq S_{\mathcal{ID}}$. More formally, let Tr : $T \neq \emptyset$, and the corresponding probability $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Tr}]$.

- Position with distinct indices for each key: Recall in our construction, each key for identity id consists of λ different 'indexed' keys, where the λ indices are chosen uniformly at random from $[2n_{\text{bd}}^2]$. Let $\mathsf{Dist\text{-}Indx}$ be the event that there exists $i \in [\lambda]$ such that the i^{th} index of each key is distinct.

- $\mathsf{Dist\text{-}Indx}_i$: Position i is the first position such that the i^{th} index of each key is distinct. Hence, by definition, $\mathsf{Dist\text{-}Indx} = \bigcup_i \mathsf{Dist\text{-}Indx}_i$.

- Tracing without correctness in i^{th} iteration: Let T_i denote the set of (identity, signatute) pairs traced in the i^{th} iteration. The event Tr_i happens if T_i is non-empty.

- Tracing with same signature as that received in key: Let T_i denote the set of (identity, signature) pairs that are traced in the i^{th} iteration. The event $\mathsf{Cor\text{-}Tr\text{-}Sig}_i$ happens if for all $(\mathsf{id}, \sigma) \in T_i$, id was queried during the key query phase, and the key generation oracle output $\mathsf{sk} \leftarrow \mathsf{Ind.KeyGen}(\mathsf{msk}_i, (\mathsf{id}, \sigma), j)$ for some index j.

- The flag is set to 'Yes' in the i^{th} iteration: The event $\mathsf{Found\text{-}Good\text{-}r}_i$ happens if in the i^{th} iteration, the flag is set to 'Yes'.
- Good decoder $\widetilde{D}$ during the $\mathsf{SubTrace}$ routine execution in i^{th} iteration: We say that event $\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i$ happens if in the i^{th} iteration, the execution reaches step 3 (that is, it found a 'good' r in the i^{th} iteration), and the decoder $\widetilde{D}$ constructed is an $\epsilon/4$ good decoder for distinguishing messages $m_0 \oplus r$ and $m_1 \oplus r$. Note that if no good r is found in step 3, then we say that $\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i$ did not happen.

With these events defined, we will now show that $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$. The proof will proceed via a series of inequalities as shown below.

$$\begin{aligned}
&\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) && (2)\\
&\geq \Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}_1(\lambda) && (3)\\
&\geq \Pr\left[\mathsf{Tr} \wedge \mathsf{Dist\text{-}Indx}\right] - \mathsf{negl}_1(\lambda) && (4)\\
&= \sum_i \Pr\left[\mathsf{Tr} \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}_1(\lambda) && (5)\\
&\geq \sum_i \Pr\left[\mathsf{Cor\text{-}Tr\text{-}Sig}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}_1(\lambda) && (6)\\
&\geq \sum_i \Pr\left[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}_2(\lambda) && (7)\\
&\geq \sum_i \Pr\left[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}_2(\lambda) && (8)\\
&\geq \sum_i \Pr\left[\mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}_3(\lambda) && (9)\\
&\geq \sum_i \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}_4(\lambda) && (10)\\
&= \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}\right] - \mathsf{negl}_4(\lambda) && (11)\\
&\geq \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}\right] - \mathsf{negl}_5(\lambda) && (12)
\end{aligned}$$

We will now discuss each of these steps, and why the above inequalities hold. Step 2 to 3 follows from the false trace guarantee. Using the false trace guarantee, we can show that $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \mathsf{negl}(\lambda)$.

Claim 9.1. $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \mathsf{negl}(\lambda)$.

Proof. From the definition of the events, it follows that $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda)$. Next, using the false trace guarantees, we can argue that $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \mathsf{negl}(\lambda)$. □

Step 3 to 4 follows from definition of $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda)$. Step 4 to 5 follows from the fact that $\mathsf{Dist\text{-}Indx} = \cup_i \mathsf{Dist\text{-}Indx}_i$, and these events are mutually exclusive. Step 5 to 6 holds because the event $\mathsf{Cor\text{-}Tr\text{-}Sig}_i$ implies Tr (we are assuming our signature scheme is perfectly correct, and hence any signature generated by the signing key is accepted by the verification algorithm). Step 6 to 7 follows from the correct-trace guarantee of the underlying $\mathsf{TT\text{-}ind}$ scheme. This is formalized in the following claim.

Claim 9.2. Assuming the traitor tracing scheme satisfies the correct-trace guarantee as defined in Definition B.2, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $i \in [\lambda]$,

$$\Pr\left[\mathsf{Cor\text{-}Tr\text{-}Sig}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] \geq \Pr\left[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathsf{negl}(\lambda)$$

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr\left[\mathsf{Good}\text{-}\widetilde{\mathsf{D}}_i \wedge \mathsf{Admissible}\text{-}\mathsf{Adv} \wedge \mathsf{Dist}\text{-}\mathsf{Indx}_i\right] - \Pr\left[\mathsf{Cor}\text{-}\mathsf{Tr}\text{-}\mathsf{Sig}_i \wedge \mathsf{Dist}\text{-}\mathsf{Indx}_i\right]$ is non-negligible. We will use $\mathcal{A}$ to build a PPT algorithm $\mathcal{B}$ that breaks the security of $\mathsf{TT}\text{-}\mathsf{ind}$. The reduction algorithm first receives the bound n_{bd} (in unary) from the adversary. It sets $n_{\mathrm{indx}} = 2 \cdot n_{\mathrm{bd}}^2$, sends n_{indx} to the challenger (in unary) and receives the public key pk_i (and key_i if $\mathsf{TT}\text{-}\mathsf{ind}$ is a public tracing scheme). For each $k \neq i$, it chooses $(\mathsf{pk}_k, \mathsf{msk}_k, \mathsf{key}_k) \leftarrow \mathsf{Ind.Setup}(1^\lambda, 1^{\kappa+\ell_{\mathsf{ss}}(\lambda)}, n_{\mathrm{indx}})$, chooses a signing/verification key, and sends the public key to $\mathcal{A}$ (and the tracing key if applicable). Next, the adversary queries for secret keys. For each query id, the reduction algorithm chooses λ indices $j_{\mathsf{id},1}, \ldots, j_{\mathsf{id},\lambda}$. It computes a signature σ on id and queries for a $\mathsf{TT}\text{-}\mathsf{ind}$ key sk_i corresponding to (id, σ) for index $j_{\mathsf{id},i}$. For $k \neq i$, it computes a sk_k using msk_k. Finally, it sends $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\lambda)$ to $\mathcal{A}$. After at most n_{bd} key queries, the adversary submits a decoding box D and messages m_0, m_1. The reduction algorithm first checks if, for all key queries, the i^{th} position's indices are distinct. If not, it outputs a dummy decoding box and quits. Next, it runs $\mathsf{isGoodDecoder}((\mathsf{pk}_i)_{i\in[\lambda]}, 1^y, m_0, m_1, r)$ for uniformly random and independently chosen r, until it finds an r s.t. $\mathsf{isGoodDecoder}$ outputs 'Yes'. If no such r is found after $\lambda \cdot y$ iterations, it quits (and outputs a dummy decoding box). It constructs $\widetilde{D}$ using D (as described in $\mathsf{SubTrace}$) and sets the distinguishing messages to be $\widetilde{m_0} = m_0 + r$, $\widetilde{m_1} = m_1 + r$. It sends $\widetilde{D}, \widetilde{m_0}, \widetilde{m_1}$ to the challenger.

First, let us compute the probability that the reduction algorithm outputs a $1/4y$ good decoding box for $\widetilde{m_0}$, $\widetilde{m_1}$. This happens if all the following events happen:

- All the position i indices of the keys are distinct.
- The decoder $\widetilde{D}$ is a $1/4y$ good decoder for $\widetilde{m_0}, \widetilde{m_1}$. This happens only if $\mathsf{isGoodDecoder}$ outputs 'Yes' for one of the randomly chosen r values.

The probability that $\mathcal{B}$ outputs a good decoding box is $\Pr\left[\mathsf{Good}\text{-}\widetilde{\mathsf{D}}_i \wedge \mathsf{Admissible}\text{-}\mathsf{Adv} \wedge \mathsf{Dist}\text{-}\mathsf{Indx}_i\right]$. Using the security of $\mathsf{TT}\text{-}\mathsf{ind}$, $\Pr\left[\mathsf{Cor}\text{-}\mathsf{Tr}\text{-}\mathsf{Sig}_i \wedge \mathsf{Dist}\text{-}\mathsf{Indx}_i\right] \geq \Pr\left[\mathsf{Good}\text{-}\widetilde{\mathsf{D}}_i \wedge \mathsf{Dist}\text{-}\mathsf{Indx}_i\right] - \mathsf{negl}(\lambda)$. Here, note that the set of (identity, signature) pairs output by the tracing algorithm is a subset of the (identity, signature) queries made by the reduction algorithm. This concludes our proof. □

Step 7 to 8 follows directly from the definition of the events. Next, we will show that $\mathsf{Found}\text{-}\mathsf{Good}\text{-}\mathsf{r}_i$ and $\overline{\mathsf{Good}\text{-}\widetilde{\mathsf{D}}_i}$ happens with negligible probability. This justifies the step 8 to 9 inequality.

Claim 9.3. There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any index i, $\Pr\left[\mathsf{Found}\text{-}\mathsf{Good}\text{-}\mathsf{r}_i \wedge \overline{\mathsf{Good}\text{-}\widetilde{\mathsf{D}}_i}\right] \leq \mathsf{negl}(\lambda)$.

Proof. Using Chernoff bounds, we can argue that $\Pr\left[\mathsf{Found}\text{-}\mathsf{Good}\text{-}\mathsf{r}_i \mid \overline{\mathsf{Good}\text{-}\widetilde{\mathsf{D}}_i}\right] < \mathsf{negl}(\lambda)$, and hence the claim follows.

□

We will now show that for every i, $\mathsf{Good}\text{-}\mathsf{Decoder}$ and $\overline{\mathsf{Found}\text{-}\mathsf{Good}\text{-}\mathsf{r}_i}$ happen with negligible probability. Hence, we can justify the step 9 to 10 transition.

Claim 9.4. There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any index i, $\Pr\left[\overline{\mathsf{Found}\text{-}\mathsf{Good}\text{-}\mathsf{r}_i} \wedge \mathsf{Good}\text{-}\mathsf{Decoder}\right] \leq \mathsf{negl}(\lambda)$.

Proof. As in the previous proof, this also follows via a simple application of Chernoff bounds, since the term $\Pr\left[\overline{\mathsf{Found}\text{-}\mathsf{Good}\text{-}\mathsf{r}_i} \mid \mathsf{Good}\text{-}\mathsf{Decoder}\right]$ can be bounded by $\mathsf{negl}(\lambda)$.

□

Next, since $\cup_i \mathsf{Dist}\text{-}\mathsf{Indx}_i = \mathsf{Dist}\text{-}\mathsf{Indx}$, and all the $\mathsf{Dist}\text{-}\mathsf{Indx}_i$ are mutually exclusive, step 10 to 11 follows. Finally, using the following claim, we can argue the step 11 to 12 inequality.

Claim 9.5. There exist a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{Good}\text{-}\mathsf{Decoder} \wedge \mathsf{Admissible}\text{-}\mathsf{Adv} \wedge \mathsf{Dist}\text{-}\mathsf{Indx}\right] \geq \Pr\left[\mathsf{Good}\text{-}\mathsf{Decoder} \wedge \mathsf{Admissible}\text{-}\mathsf{Adv}\right] - \mathsf{negl}(\lambda).$$

Proof. Consider the following experiment: choose $x_{i,j} \leftarrow [2 \cdot n_{\mathrm{bd}}^2]$ for each $i \in [\lambda]$, $j \in [n_{\mathrm{bd}}]$ independently. Let $Y_i = 1$ if the set $\{x_{i,j}\}_{j\in[\lambda]}$ consists of n_{bd} distinct entries, else $Y_i = 0$. Using a simple union bound, it follows that $\Pr[Y_i = 0] \leq 1/4$, and since all Y_i are independent, $\Pr[\ \forall i,\ Y_i = 0] \leq 1/4^{\lambda}$.

From the above experiment, it follows that $\Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \overline{\mathsf{Dist\text{-}Indx}}] \leq \mathsf{negl}(\lambda)$. As a result,

$$\begin{aligned}
&\Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}\right]\\
&= \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}\right] + \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \overline{\mathsf{Dist\text{-}Indx}}\right]\\
&\leq \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}\right] + \mathsf{negl}(\lambda)
\end{aligned}$$

□

□

10 Unbounded (Full) Embedded-Identity Traitor Tracing

We will now present the syntax and definitions for general traitor tracing with embedded identities.

Let $\mathcal{T}$ be a (unbounded keys, public/private tracing)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$. It consists of five algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^\kappa) \rightarrow (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$: The setup algorithm takes as input the security parameter λ, identity space index κ and outputs a master secret key msk and a public key pk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa) \rightarrow \mathsf{sk}_{\mathsf{id}}$: The key generation algorithm takes as input the master secret key and identity $\mathsf{id} \in \{0,1\}^\kappa$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \rightarrow \mathsf{ct}$: The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow z$: The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, Q_{\mathrm{bd}}, m_0, m_1) \rightarrow T \subseteq \{0,1\}^\kappa$. The trace algorithm has oracle access to a program D, it takes as input a master secret key key, parameters y and Q_{bd}, and two messages m_0, m_1. It outputs a set T of identities, where $T \subseteq \{0,1\}^\kappa$.

Correctness. A traitor tracing scheme is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$ and identity $\mathsf{id} \in \{0,1\}^\kappa$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Efficiency Let $\mathsf{T\text{-}s}, \mathsf{T\text{-}e}, \mathsf{T\text{-}k}$ be functions. A (bounded keys, public/private tracing)-embedded identity tracing scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$)-efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa)$ is at most $\mathsf{T\text{-}s}(\lambda, \kappa)$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda, \kappa)$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa)$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa)$.
- The number of oracle calls made by $\mathsf{Trace}^D(\mathsf{key}, 1^y, Q_{\mathrm{bd}}, m_0, m_1)$ to decoding box D is at most $\mathsf{T\text{-}t}(\lambda, \kappa, y, Q_{\mathrm{bd}})$.
- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa)$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa)$.

10.1 Security

The $\mathsf{IND\text{-}CPA}$ security definition is identical to the one in previous sections; the tracing-based security definition is very similar, but there is no bound on the number of secret key queries.

Definition 10.1 (Secure tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (unbounded keys, public/private tracing)-embedded identity tracing scheme. For any non-negligible function $\epsilon(\cdot)$, polynomial $p(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT\text{-}emb}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Figure 7.

Experiment $\mathsf{Expt\text{-}TT\text{-}emb}^{\mathcal{T}}_{\mathcal{A},\epsilon,p}(\lambda)$

- $1^\kappa \leftarrow \mathcal{A}(1^\lambda)$.
- $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{key}, 1^{1/\epsilon(\lambda)}, p(\lambda), m_0, m_1)$.

Let $S_{\mathcal{ID}}$ be the set of identities queried by $\mathcal{A}$. Here, $O(\cdot)$ is an oracle that has msk hardwired, takes as input an identity $\mathsf{id} \in \{0,1\}^\kappa$ and outputs $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$.

Figure 7: Experiment $\mathsf{Expt\text{-}TT\text{-}emb}$

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon, p$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$
  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon,p}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder} \wedge p(\lambda) \geq |S_{\mathcal{ID}}|]$.
- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathcal{ID}}$
  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.
- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S_{\mathcal{ID}}$
  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A traitor tracing scheme $\mathcal{T}$ is said to be secure if for every PPT adversary $\mathcal{A}$, polynomials $q(\cdot)$, $p(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathsf{negl}_1(\cdot)$, $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathsf{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon,p}(\lambda) - \mathsf{negl}_2(\lambda).$$

Remark 10.1. Note that unlike Definitions 9.1 and B.2, here the trace algorithm takes an additional parameter Q_{bd}. In the correct trace definition, we require that as long as the tracing algorithm uses a bound greater than the number of keys queried, the tracing algorithm must identify at least one traitor. However, the false trace guarantee should hold for all polynomially bounded Q_{bd} values. In particular, even if the number of keys queried is more than the bound used in tracing, the trace algorithm must not output an identity that was not queried. We can show that this definition implies the 'standard' tracing definition where the trace algorithm does not take this bound as input. One simply needs to run this bounded-version of trace with increasing powers of two until the trace algorithm outputs at least one traitor.

The tracing algorithm in [NWZ16] also requires a bound q. However, in their definition, the false trace and correct trace events are defined only when the tracing bound is equal to the number of keys queried. As a result, it is not clear if this definition can be used to achieve the 'standard' tracing definition.

10.2 Going from Bounded to Unbounded

In this section, we provide an efficient generic transformation that removes the bound set on the number of users/keys that an adversary is allowed to corrupt during setup.

10.2.1 Construction

Let $\mathsf{TT\text{-}bd} = (\mathsf{BD.Setup}, \mathsf{BD.KeyGen}, \mathsf{BD.Enc}, \mathsf{BD.Dec}, \mathsf{BD.Trace})$ be a (bounded keys, public/private tracing)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$, identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$, and with ($\mathsf{T\text{-}setup}$, $\mathsf{T\text{-}enc}$, $\mathsf{T\text{-}key}$)-efficiency. We use $\mathsf{TT\text{-}bd}$ to build a traitor tracing $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ with unbounded number of users as follows. (Here we provide a transformation for TT schemes with secret key tracing, but the construction can be easily extended to work in the public tracing setting as well.)

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the $\mathsf{TT\text{-}bd}$ setup algorithm λ times with increasing values of the user bound n_{bd} as follows:

$$\forall i \in [\lambda], \quad (\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{BD.Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}} = 2^i).$$

It then sets the master secret and public keys as an λ-tuple of all these keys, i.e. $\mathsf{msk} = (\mathsf{msk}_i)_{i \in [\lambda]}$, $\mathsf{pk} = (\mathsf{pk}_i)_{i \in [\lambda]}$ and $\mathsf{key} = (\mathsf{pk}_i, \mathsf{key}_i)_{i \in [\lambda]}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}$. Let $\mathsf{msk} = (\mathsf{msk}_i)_{i \in [\lambda]}$. The key generation algorithm runs the $\mathsf{TT\text{-}bd}$ key generation algorithm with all λ keys independently as $\mathsf{sk}_i \leftarrow \mathsf{BD.KeyGen}(\mathsf{msk}_i, \mathsf{id})$ for $i \in [\lambda]$. It outputs the secret key sk as $\mathsf{sk} = (\mathsf{sk}_i)_{i \in [\lambda]}$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. Let $\mathsf{pk} = (\mathsf{pk}_i)_{i \in [\lambda]}$. The encryption algorithm first chooses $\lambda - 1$ random messages as $r_i \leftarrow \mathcal{M}$ for $i \in [\lambda - 1]$. Next, it sets $r_\lambda = m \oplus \left(\bigoplus_{i=1}^{\lambda-1} r_i\right)$. It then encrypts messages r_i under key pk_i as follows:

$$\forall i \in [\lambda], \quad \mathsf{ct}_i \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_i, r_i).$$

Finally, it outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_i)_{i \in [\lambda]}$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. Let $\mathsf{sk} = (\mathsf{sk}_i)_{i \in [\lambda]}$, and $\mathsf{ct} = (\mathsf{ct}_i)_{i \in [\lambda]}$. The decryption algorithm runs the $\mathsf{TT\text{-}bd}$ decryption on each secret key-ciphertext pair as $z_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_i)$ for $i \in [\lambda]$.

If $z_i = \perp$ for any $i \in [\lambda]$, then it outputs $z = \perp$, otherwise it outputs $z = \bigoplus_{i=1}^{\lambda} z_i$ as the message.

```
$\mathsf{isGoodDecoder}^D((\mathsf{pk}_i), 1^y, m_0, m_1, r, i)$

Input: Public keys $(\mathsf{pk}_i)_{i \in [\lambda]}$, Parameter $y$, Messages $m_0, m_1, r$, Index $i \in [\lambda]$.
Output: Yes/No.
  1. Set $\mathsf{count} = 0$. (Let $\epsilon = 1/y$.)
  2. For $j = 1$ to $\lambda \cdot y$:
       - Choose $\lambda - 1$ messages $r_k$ randomly for $k \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{k \in [\lambda] \setminus \{i\}} r_k = r$. (That is,
         bit-wise parity of the messages chosen matches the message $r$.)
       - Choose random bit $b \leftarrow \{0,1\}$, and compute ciphertexts as $\mathsf{ct}_k \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_k, r_k)$ for $k \in [\lambda] \setminus \{i\}$,
         and $\mathsf{ct}_i \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_i, r \oplus m_b)$.
       - Query ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda)$ to the oracle $D$. Let $b'$ denote the oracle's response.
       - If $b = b'$, set $\mathsf{count} = \mathsf{count} + 1$.
  3. If $\mathsf{count}/(\lambda \cdot y) \geq 1/2 + \epsilon/3$, then output 'Yes'. Otherwise output 'No'.
```

Figure 8: Routine $\mathsf{isGoodDecoder}$

$\mathsf{Trace}^D(\mathsf{key}, 1^y, Q_{\mathrm{bd}}, m_0, m_1) \to T$. Let $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i \in [\lambda]}$ and $\epsilon = 1/y$. First we define a supplementary algorithms $\mathsf{isGoodDecoder}$ (in Figure 8) and $\mathsf{SubTrace}$ (in Figure 9) that both get oracle access to the decoder D and take as input all λ tracing/public key pairs $(\mathsf{key}_i, \mathsf{pk}_i)_i$, parameter y, messages m_0, m_1, r and an index $i \in [\lambda]$. The tracing algorithm executes the following procedure using $\mathsf{isGoodDecoder}$ and $\mathsf{SubTrace}$ routines as follows:

$\mathsf{SubTrace}^D(\mathsf{key}, 1^y, m_0, m_1, r, i)$

Input: Keys $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: $T \subseteq \{0,1\}^\kappa$.

1. It runs the $\mathsf{TT\text{-}bd}$ tracing algorithm on inputs — keys $\mathsf{key}_i, \mathsf{pk}_i$, parameter $4y$, messages $m_0 \oplus r, m_1 \oplus r$ — and with oracle access to oracle $\widetilde{D}$ which we define next.
2. On each query ct to oracle $\widetilde{D}$ by $\mathsf{BD.Trace}$, the sub-tracing algorithm first chooses $\lambda - 1$ messages r_j randomly for $j \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r$. (That is, bit-wise parity of the messages chosen matches the message r.) It encrypts these messages as $\mathsf{ct}_j \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_j, r_j)$, and then sends the ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda)$ to the oracle D as its query. And it forwards the oracle D's response to the $\mathsf{BD.Trace}$ algorithm.
3. Finally, the $\mathsf{BD.Trace}$ algorithm outputs a set T. The sub-tracing algorithm outputs the same set T as its output.

In short, $\mathsf{SubTrace}^D(\mathsf{key}, 1^y, m_0, m_1, r, i) = \mathsf{BD.Trace}^{\widetilde{D}}((\mathsf{key}_i, \mathsf{pk}_i), 1^{4y}, m_0 \oplus r, m_1 \oplus r)$,

$$\widetilde{D}(\mathsf{ct}) = D(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda),$$
$$\text{where} \quad \forall j \in [\lambda] \setminus \{i\}, \; r_j \leftarrow \mathcal{M} \text{ such that } \textstyle\bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r,$$
$$\forall j \in [\lambda] \setminus \{i\}, \; \mathsf{ct}_j \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_j, r_j)$$

Figure 9: Routine $\mathsf{SubTrace}$

1. Set $i = \lceil \log Q_{\mathrm{bd}} \rceil$.
2. Set flag = 'No'. For $j = 1$ to $\lambda \cdot y$:
   - Choose a random message $r \leftarrow \mathcal{M}$.
   - Run $\mathsf{isGoodDecoder}$ as $\mathsf{flag} \leftarrow \mathsf{isGoodDecoder}^D((\mathsf{pk}_j), 1^y, m_0, m_1, r, i)$.
   - If flag = 'Yes', break. Else, continue.
3. If flag = 'Yes', run $\mathsf{SubTrace}$ as $T \leftarrow \mathsf{SubTrace}^D(\mathsf{key}, 1^y, m_0, m_1, r, i)$. Else, set $T = \emptyset$.
4. Output T.

Next, we prove the following.

Theorem 10.1. If $\mathsf{TT\text{-}bd} = (\mathsf{BD.Setup}, \mathsf{BD.KeyGen}, \mathsf{BD.Enc}, \mathsf{BD.Dec}, \mathsf{BD.Trace})$ is a secure (bounded keys, public/private tracing)-embedded identity tracing scheme (as per Definitions 9.1 and 4.2) with ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)-efficiency, then the scheme $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ (described in Section 10.2.1) is a secure (unbounded keys, public/private tracing)-embedded identity tracing scheme (as per Definitions 10.1 and 4.2) with ($\mathsf{T\text{-}s}'$, $\mathsf{T\text{-}e}'$, $\mathsf{T\text{-}k}'$, $\mathsf{T\text{-}d}'$, $\mathsf{T\text{-}t}'$, $\mathsf{S\text{-}c}'$, $\mathsf{S\text{-}k}'$)-efficiency, where the efficiency measures are related as follows:

- $\mathsf{T\text{-}s}'(\lambda, \kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}s}(\lambda, \kappa, 2^i)$,
- $\mathsf{T\text{-}k}'(\lambda, \kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}k}(\lambda, \kappa, 2^i)$,
- $\mathsf{T\text{-}e}'(\lambda, \kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}e}(\lambda, \kappa, 2^i) + \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}d}'(\lambda, \kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}d}(\lambda, \kappa, 2^i) + \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}t}'(\lambda, \kappa, y, Q_{\mathrm{bd}}) = \mathsf{T\text{-}t}(\lambda, \kappa, 2^{\lceil \log Q_{\mathrm{bd}} \rceil}, 4y) + \lambda^2 \cdot y^2$,
- $\mathsf{S\text{-}c}'(\lambda, \kappa) = \sum_{i=1}^{\lambda} \mathsf{S\text{-}c}(\lambda, \kappa, 2^i)$,
- $\mathsf{S\text{-}k}'(\lambda, \kappa) = \sum_{i=1}^{\lambda} \mathsf{S\text{-}k}(\lambda, \kappa, 2^i)$.

Proof. **Correctness**: Fix any security parameter λ, public key $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$, master secret key $\mathsf{msk} = \left((\mathsf{msk}_i)_{i\in[\lambda]}\right)_{i\in[\lambda]}$, tracing key $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$, message $m \in \mathcal{M}$ and identity id. The encryption algorithm chooses $\{r_i\}_{i\in[\lambda]}$ such that $\bigoplus_i r_i = m$, computes $\mathsf{ct}_i \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_i, r_i)$ and sets $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$. The key generation algorithm outputs λ keys $(\mathsf{sk}_i)_{i\in[\lambda]}$, where sk_i is a key for identity id computed using msk_i. From the correctness of the underlying scheme $\mathsf{TT\text{-}bd}$, it follows that decryption of ct_i using sk_i outputs r_i. As a result, the decryption of ct using sk outputs message m.

IND-CPA security: $\mathsf{IND\text{-}CPA}$ security of our scheme follows directly from the $\mathsf{IND\text{-}CPA}$ security of $\mathsf{TT\text{-}bd}$. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks the $\mathsf{IND\text{-}CPA}$ security of our scheme with advantage ϵ. We can use $\mathcal{A}$ to break the $\mathsf{IND\text{-}CPA}$ security of $\mathsf{TT\text{-}bd}$ with advantage ϵ. The reduction algorithm first sends λ and bound $= 2$ to the challenger, and receives pk_1 from the $\mathsf{TT\text{-}ind}$ challenger. It chooses $(\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{BD.Setup}(1^\lambda, 1^\kappa, 2^i)$ for each $i \in \{2, \ldots, \lambda\}$, and sends $(\mathsf{pk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. (If $\mathsf{TT\text{-}bd}$ is a public tracing scheme, then the reduction algorithm also receives a tracing key key_1 from the challenger, and it sends the tracing key $(\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$).

Next, $\mathcal{A}$ sends two messages m_0, m_1 to $\mathcal{B}$. The reduction algorithm chooses $r_2, \ldots, r_n \leftarrow \mathcal{M}$, sets $m'_b = m_b \oplus \left(\bigoplus_{i>1} r_i\right)$ and sends (m'_0, m'_1) to the challenger. The challenger sends ct_1 to $\mathcal{B}$. The reduction algorithm computes encryptions of $r_2, \ldots, r_\lambda$, and sends $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. The adversary sends its guess, which the reduction algorithm forwards to the challenger.

Clearly, if $\mathcal{A}$ has advantage ϵ, then so does $\mathcal{B}$.

Correct Trace and False Trace guarantees: We will show that our scheme satisfies Definition 10.1.

False Trace: First, let us consider the false trace probability. False trace happens if the sub-tracing algorithm outputs a non-empty set T such that $T \not\subseteq S_{\mathcal{ID}}$, where $S_{\mathcal{ID}}$ is the set of keys queried by the adversary. We will show that if there exists an adversary $\mathcal{A}$, polynomial p and non-negligible functions ϵ, η such that $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \geq \eta(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the false-trace guarantee of $\mathsf{TT\text{-}bd}$.

The reduction algorithm $\mathcal{B}$ first sends $p(\lambda)$ (in unary) to the $\mathsf{TT\text{-}bd}$ challenger, and let $i = \lceil p(\lambda) \rceil$. It receives a public key pk_i (and a tracing key if it is a public tracing scheme). It chooses $(\mathsf{msk}_j, \mathsf{pk}_j, \mathsf{key}_j)$ for all $j \neq i$, and sends $\left(\mathsf{pk}_j\right)$ to $\mathcal{A}$ (and $\left(\mathsf{key}_j\right)$ for a public tracing scheme). Next, the adversary requests for keys. For each query id, the reduction algorithm computes $\mathsf{sk}_{\mathsf{id},j}$ using msk_j if $j \neq i$. It sends id to the challenger, and receives $\mathsf{sk}_{\mathsf{id},i}$. It sends $\mathsf{sk}_{\mathsf{id}} = (\mathsf{sk}_{\mathsf{id},j})$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a decoding box D and messages m_0, m_1. The reduction algorithm first runs $\mathsf{isGoodDecoder}(\left\{\mathsf{pk}_j\right\}, 1^y, m_0, m_1, r, i)$ for $\lambda \cdot y$ choices of r until it finds an r s.t. $\mathsf{isGoodDecoder}$ outputs 'Yes'. The reduction algorithm outputs $\widetilde{D}$ (as defined in $\mathsf{SubTrace}$) as the decoding box, and $m_0 \oplus r$, $m_1 \oplus r$ as the two messages.

Clearly, if $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \geq \eta(\lambda)$, then $\mathcal{B}$ breaks the false-trace guarantee of $\mathsf{TT\text{-}bd}$.

Correct Trace:

As before, our proof will proceed via a sequence of inequalities. The events are defined exactly as in the proof of Theorem 9.1. Let $i = \lceil p(\lambda) \rceil$.

$$
\begin{aligned}
&\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) && (13)\\
&= \Pr[\mathsf{Cor\text{-}Tr}_i] && (14)\\
&\geq \Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathcal{ID}}|] - \mathsf{negl}_1(\lambda) && (15)\\
&\geq \Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathcal{ID}}| \wedge \mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder}] && (16)\\
&\geq \Pr[\mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder} \wedge p(\lambda) \geq |S_{\mathcal{ID}}|] && (17)\\
&\geq \Pr[\mathsf{Good\text{-}Decoder} \wedge p(\lambda) \geq |S_{\mathcal{ID}}|] && (18)
\end{aligned}
$$

The first equality (Step 13 to 14) follows from the definition of corret-trace. Next, Step 14 to 15 follows from the correct-trace guarantee of $\mathsf{TT\text{-}bd}$. This is formalized in the following claim.

Claim 10.1. Assuming $\mathsf{TT\text{-}bd}$ satisfies Definition 9.1, for any PPT adversary $\mathcal{A}$, polynomial $p(\cdot)$ and non-negligible function $\epsilon(\cdot)$, $\Pr[\mathsf{Cor\text{-}Tr}_i] \geq \Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathcal{ID}}|] - \mathsf{negl}_1(\lambda)$.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, polynomial $p(\cdot)$ and non-negligible functions $\epsilon(\cdot), \eta(\cdot)$ such that $\Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathcal{ID}}|] - \Pr[\mathsf{Cor\text{-}Tr}_i] \geq \eta(\lambda)$. We will use $\mathcal{A}$ to construct a PPT algorithm $\mathcal{B}$ that breaks the correct-trace guarantee of $\mathsf{TT\text{-}bd}$.

Let $i = \lceil \log p(\lambda) \rceil$. The reduction algorithm $\mathcal{B}$ sends $\lambda, p(\lambda)$ (in unary) to the challenger, and receives pk_i (and the tracing key key_i if $\mathsf{TT\text{-}bd}$ is a public tracing scheme). It then chooses $(\mathsf{msk}_j, \mathsf{pk}_j, \mathsf{key}_j) \leftarrow \mathsf{BD.Setup}(1^\lambda, 1^\kappa, 2^j)$ for all $j \neq i$ and sends $\left(\mathsf{pk}_j\right)_{j \in [\lambda]}$. It then receives key queries (at most $p(\lambda)$ key queries). For each key query id, the reduction algorithm generates $\mathsf{sk}_{\mathsf{id},j}$ for $j \neq i$, sends id to the challenger, and receives $\mathsf{sk}_{\mathsf{id},i}$. It sends $\mathsf{sk}_{\mathsf{id}} = (\mathsf{sk}_{\mathsf{id},j})$ to the adversary.

Finally, the adversary sends a decoding box D and messages m_0, m_1. The reduction algorithm runs $\mathsf{isGoodDecoder}$ with different r values until the flag is 'Yes'. Then, it uses that r value to set the decoding box $\widetilde{D}$ (as in $\mathsf{SubTrace}$), messages $m'_b = m_b \oplus r$ and sends $\widetilde{D}, m'_0, m'_1$ to the challenger. □

The inequality from 15 to 16 follows directly from the definition of the events. Next, the justification for transition from Step 16 to 17 and Step 17 to 18 is similar to the proofs of Claim 9.3 and Claim 9.4 respectively.

□

References

[ABP+17] Shweta Agrawal, Sanjay Bhattacherjee, Duong Hieu Phan, Damien Stehlé, and Shota Yamada. Efficient public trace and revoke from standard assumptions: Extended abstract. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2277–2293, 2017.

[ADM+07] Michel Abdalla, Alexander W. Dent, John Malone-Lee, Gregory Neven, Duong Hieu Phan, and Nigel P. Smart. Identity-based traitor tracing. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, pages 361–376, 2007.

[BF99] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 338–353, 1999.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 533–556, 2014.

[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 *CRYPTO*, pages 1–18, 2001.

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

[BHJ+13] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. Practical signatures from standard assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 461–485. Springer, 2013.

[BP08] Olivier Billet and Duong Hieu Phan. Efficient traitor tracing from collusion secure codes. In *Information Theoretic Security, Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings*, pages 171–182, 2008.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 211–220, 2006.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 480–499, 2014.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[CFNP00] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Trans. Information Theory*, 46(3):893–910, 2000.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, 2015.

[CPP05] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 542–558, 2005.

[CVW$^+$18] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from lwe made simple and attribute-based. In *TCC*, 2018.

[DS15] Nico Döttling and Dominique Schröder. Efficient pseudorandom functions via on-the-fly adaptation. In *Annual Cryptology Conference*, pages 329–350. Springer, 2015.

[FNP07] Nelly Fazio, Antonio Nicolosi, and Duong Hieu Phan. Traitor tracing with optimal transmission rate. In *Information Security, 10th International Conference, ISC 2007, Valparaíso, Chile, October 9-12, 2007, Proceedings*, pages 71–88, 2007.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 44–61, Berlin, Heidelberg, 2010. Springer-Verlag.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, 2015.

[GKRW18] Rishab Goyal, Venkata Koppula, Andrew Russell, and Brent Waters. Risky traitor tracing and new differential privacy negative results. In *Crypto*, 2018.

[GKSW10] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 121–130, New York, NY, USA, 2010. ACM.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *STOC*, 2018.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[KD98] Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 145–157, 1998.

[KY02a] Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 450–465, 2002.

[KY02b] Kaoru Kurosawa and Takuya Yoshida. Linear code implies public-key traitor tracing. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, pages 172–187, 2002.

[LPSS14] San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-lwe and applications in traitor tracing. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 315–334, 2014.

[NWZ16] Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 388–419, 2016.

[PST06] Duong Hieu Phan, Reihaneh Safavi-Naini, and Dongvu Tonien. Generic construction of hybrid public key traitor tracing with full-public-traceability. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 264–275, 2006.

[SSW01] Jessica Staddon, Douglas R. Stinson, and Ruizhong Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Information Theory*, 47(3):1042–1049, 2001.

[SW98] Douglas R. Stinson and Ruizhong Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discrete Math.*, 11(1):41–53, 1998.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 600–611, 2017.

A Preliminaries (Cont'd)

A.1 Signature Schemes

A signature scheme $\mathcal{S} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ with message space $\mathcal{M}$ consists of three algorithms, as follows:

$\mathsf{Setup}(1^\lambda)$ is a randomized algorithm that takes security parameter λ as input and returns a pair of keys $(\mathsf{sk}, \mathsf{vk})$, where sk is the signing key and vk is the verification key.

$\mathsf{Sign}(\mathsf{sk}, m)$ is a possibly randomized algorithm that takes as input the signing key sk, and a message m, and returns a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ is a determistic algorithm that takes as input the verification key vk, a message m, and a signature σ, and outputs 1 (accepts) if verification succeeds, and 0 (rejects) otherwise.

Correctness : A signature scheme $\mathcal{S}$ must satisfy the following correctness requirement: For all $\lambda \in \mathbb{N}$, $m \in \mathcal{M}$, and signing/verification keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$

$$\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1.$$

Definition A.1. A signature scheme $\mathcal{S} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ is a secure signature scheme if for every PPT attacker $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{adv}^{\mathcal{S}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$, where advantage of $\mathcal{A}$ is defined as

$$\mathsf{adv}^{\mathcal{S}}_{\mathcal{A}}(\lambda) = \Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 : \begin{array}{c} (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m^*, \sigma^*) = \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right],$$

and $\mathcal{A}$ should never have queried m^* to Sign oracle.

A.2 Public Key Encryption

A public key encryption scheme PKE with message space $\mathcal{M}$ consists of three algorithms Setup, Enc and Dec with the following syntax:

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$ The setup algorithm takes as input the security parameter 1^λ and outputs a public key pk and secret key sk.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}) \rightarrow \mathsf{ct}$ The encryption algorithm takes as input a public key pk and a message $m \in \mathcal{M}$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow x \in \mathcal{M} \cup \{\perp\}$ The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $x \in \mathcal{M} \cup \{\perp\}$.

Correctness: For correctness, we require that for all security parameters λ, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and messages $m \in \mathcal{M}$, $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Definition A.2 (IND-CPA Security)**.** A public key encryption scheme $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ is said to be IND-CPA secure if for all security parameters λ, stateful PPT adversaries $\mathcal{A}$, $\mathsf{adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$ is negligible in λ, where advantage of $\mathcal{A}$ is defined as

$$\mathsf{adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = \Pr\left[\mathcal{A}(\mathsf{ct}) = b : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda);\ b \leftarrow \{0,1\} \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk});\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array}\right].$$

B Traitor Tracing

The notion of traitor tracing was introduced by Chor, Fiat and Naor [CFN94]. In a traitor tracing scheme for n parties, the setup algorithm chooses a master secret key, a public key and n secret keys for the users. Encryption can be performed using the public key, and each user can decrypt the ciphertext using his/her secret key. There is also a trace algorithm that, given black box access to a successful pirate decoding box, can catch the traitors who colluded to create the pirate decoding box. The trace algorithm must catch a traitor if a pirate decoding box can distinguish between encryptions of two adversarially chosen messages.

This (traditional) definition of traitor tracing (described formally in Section B.1) has two limitations. First, the number of users is fixed during setup. Instead, we would like to have a separate key generation algorithm which uses a master secret key (generated during setup) to generate secret keys for each user. Second, we would like to embed meaningful information in the secret keys, so that if a user u is traced, then the tracing algorithm could directly output the relevant information about the user u. The recent work of Nishimaki, Wichs, and Zhandry [NWZ16] introduced a more general syntax/definitions for traitor tracing, where the traitor tracing scheme has a separate key generation algorithm that can embed meaningful information in the secret keys, and the tracing algorithm can trace this information with black-box access to a pirate decoding box. This notion is described formally in Section 10. Before describing this general notion, we provide a weaker notions (in Section 4.1) which could be useful in certain practical situations, and also work as stepping stones for achieveing the general notion.

B.1 (Traditional) Public Key Traitor Tracing

Here, we first present the definition of traitor tracing. This part is taken verbatim from [GKW18].

A traitor tracing scheme $\mathcal{T}$ with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four PPT algorithms $\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$. The setup algorithm takes as input the security parameter λ (in unary), number of users n (in unary), and outputs a master secret key msk, a public key pk and n secret keys $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

$\mathsf{Trace}^D(\mathsf{msk}, 1^y, m_0, m_1) \to T$. The trace algorithm has oracle access to a program D, it takes as input a master secret key msk, parameter y (in unary) and two messages m_0, m_1. It outputs a set $T \subset \{1, 2, \ldots, n\}$.

Correctness. Informally, correctness requirement states decrypting an encryption of message m using any one of the valid secret keys must output m. Formally, a traitor tracing scheme is said to be correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in \{1, 2, \ldots, n\}$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Definition B.1. A traitor tracing scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ is said to have *public tracing* if the tracing algorithm Trace uses the public key (instead of the master secret key).

B.1.1 Security

There are two security requirements for a traitor tracing scheme. First, it is required that it satisfies IND-CPA security. Second, it is required that the tracing algorithm must (almost always) correctly trace at least one key used to create a pirate decoding box (whenever the pirate box successfully decrypts with noticeable probability) as well as it should not falsely accuse any user of cheating. The formal definitions are provided below.

Definition B.2 (Ind-secure traitor tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a traitor tracing scheme. For any non-negligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined as follows.

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon$):

Experiment $\mathsf{Expt\text{-}TT}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^n \leftarrow \mathcal{A}(1^\lambda)$
- $(\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{msk}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$.

Here, $O(\cdot)$ is an oracle that has $\{\mathsf{sk}_i\}_{i\in[n]}$ hardwired, takes as input an index $i \in [n]$ and outputs sk_i. Let S be the set of indices queried by $\mathcal{A}$.

Figure 10: Experiment Expt-TT

- Good-Decoder : $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$
  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder}]$.
- Cor-Tr : $T \neq \emptyset \wedge T \subseteq S$
  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.
- Fal-Tr : $T \not\subseteq S$
  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A traitor tracing scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathsf{negl}_1(\cdot)$, $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathsf{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathsf{negl}_2(\lambda).$$