New Approaches to Traitor Tracing with Embedded Identities

Rishab Goyal[1], Venkata Koppula[2], and Brent Waters[3]*

[1] University of Texas at Austin, USA
rgoyal@cs.utexas.edu
[2] Weizmann Institute of Science, Israel
venkata.koppula@weizmann.ac.il
[3] University of Texas at Austin and NTT Research, USA
bwaters@cs.utexas.edu

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.

* Supported by NSF CNS-1908611, CNS-1414082, DARPA SafeWare and Packard Foundation Fellowship.

1 Introduction

Traitor tracing (TT) systems, as introduced by Chor, Fiat, and Naor [14], 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 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 [14,34,28,8,15,33,27,29,32,12,1,19,6,30,2] allowed bounded collusions; we focus on unbounded collusion [9,10,22,20,11,31,26,13]. 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.[4] 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.[5] 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 [31]. Their approach was to directly work with the existing private linear broadcast encryption (PLBE) framework [9], however that resulted in solutions based 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 [21,16,23,17], or indistinguishability obfuscation [4,5]. An important open question here is whether the above problem of embedded information traitor

[4] 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.

[5] 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.

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.[6]

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	$\lvert\mathsf{ct}\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	Tr. Mode	Unbdd.
PKE	$n \cdot \kappa \cdot \mathsf{poly}(\lambda)$	$n \cdot \kappa \cdot \mathsf{poly}(\lambda)$	$\kappa \cdot \mathsf{poly}(\lambda)$	Pub.	No
Bilinear	$\sqrt{n \cdot \kappa} \cdot \mathsf{poly}(\lambda)$	$\sqrt{n \cdot \kappa} \cdot \mathsf{poly}(\lambda)$	$\log n + \kappa + \mathsf{poly}(\lambda)$	Pub.	No
LWE	$(\log n + \kappa) \cdot \mathsf{poly}(\lambda)$	$\mathsf{poly}(\lambda)$	$(\log n + \kappa) \cdot \mathsf{poly}(\lambda)$	Priv.	Yes

Table 1. Embedded Identity Traitor Tracing. The 'Tr. Mode' column indicates whether tracing is public or private.

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

[6] Nishimaki, Wichs, and Zhandry [31] 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).

trace and revoke scheme of Boneh-Waters [10]. And, for building an LWE-based solution we adapt the recently introduced Mixed Functional Encryption (Mixed FE) schemes [26,13] 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 [26] and two of three follow-up constructions by Chen et al. [13] were proven to be only selectively-secure.) Therefore, our approach also answers the question whether adaptivity is necessary 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 [31], 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) [9]. 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.[7] 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.

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.[8] 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

[7] Traditionally, the tracing algorithm was defined to work only if the decoder box could decrypt encryptions of random messages. However, as discussed in [24], 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.

[8] 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.

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) [9]. 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 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 [9,26], 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.

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.

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 [9,10] 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[9]) 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.[10] 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 [10]. Our starting point is the trace and revoke (broadcast) scheme by Boneh and Waters (BW) [10]. 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 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.

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

[10] 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..

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 consists of the following terms: $\left\{E_x, G_x, E_q, E_{q,x}, G_{q,x}, \{H_{x,\ell,b}\}_{\ell,b}\right\}_x$, 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 $\varGamma_1$, pair K and R_x to compute $\varGamma_2$, and output $B_x \cdot \varGamma_1/\varGamma_2$ as the decryption.

The full scheme and security proof is discussed in the full version of our paper. As an alternate approach for constructing an EIPLBE scheme, one could use a functional encryption scheme for quadratic functions. Such a scheme was recently proposed by Baltico et al. [3]. However, one of the contributions of our work is to fix the BW scheme, and hence we chose to provide a direct construction for EIPLBE, based on the BW scheme. 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.

EIPLBE via Learning With Errors In a recent work, Goyal, Koppula, and Waters [26] 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 [26] showed a construction of Mixed FE for log-depth circuits. A recent work by Chen et al. [13] showed three different constructions for the same. To construct PLBE, [26] 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), [26] (and later [13]) 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 [13]). 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.[11]

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 [7,18].

2.6 Comparing Techniques

We conclude by giving some further comparisons between the techniques we introduce and those from the earlier work of NWZ [31]. The closest point for

[11] 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.

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 [31] 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 [31] requires *adaptive* security of the underlying primitive. The only known scheme from standard assumptions that can handle exponentially large identity space is the [13] which builds the core "Mixed FE" component from lockable obfuscation [25,35]. It is notable that the private constrained PRF-based construction of [13] and the earlier [26] 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 Traitor Tracing with Embedded Identities

3.1 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 $\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$ be functions. A (indexed keygen, public/private)-embedded identity tracing scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)- 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 $\mathsf{T\text{-}e}(\lambda, \kappa, n_{\text{indx}})$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa, n_{\text{indx}})$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa, n_{\text{indx}})$.
- 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 $\mathsf{T\text{-}t}(\lambda, \kappa, n_{\text{indx}}, y)$.
- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa, n_{\text{indx}})$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa, n_{\text{indx}})$.

Definition 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.*

Security As in the traditional traitor tracing definitions, we have two security definitions. The first security definition (IND-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 2 (IND-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* IND-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 probability is at most $1/2 + \mathsf{negl}(\lambda)$:*

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

Definition 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 expt. $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Figure 1.*

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

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

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

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

Each oracle query made by the adversary $\mathcal{A}$ consists of an index-identity pair $(i, \mathsf{id}) \in [n_{\rm indx}] \times \{0,1\}^{\kappa}$. 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 $i \in [n_{\rm indx}]$, the adversary is allowed to make at most one key query. However, for different indices $i, i' \in [n_{\rm indx}]$, the identity can be same (that is, (i, id) and (i', id) are valid queries if $i \neq i'$).*

Fig. 1. Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}$

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

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

Remark 1. We want to point out that in both IND-CPA and secure tracing games we require the adversary to output the index bound $n_{\rm indx}$ *in unary instead of binary* (i.e., $\mathcal{A}$ outputs $(1^{\kappa}, 1^{n_{\rm indx}})$ instead of $(1^{\kappa}, n_{\rm indx})$). 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 $n_{\rm indx}$. However, the setup algorithm is given the input $n_{\rm indx}$ *in binary.* This distinction will later be useful in our constructions and security proofs.

4 A New Framework for Embedded-Identity Traitor Tracing

4.1 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 [9] 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 a negligible function $\mathsf{negl}(\cdot)$ 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 probabilities are at least $1 - \mathsf{negl}(\lambda)$:

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

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

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 $\mathsf{S\text{-}k}(\lambda, \kappa, n)$.

q-query EIPLBE Security Now we provide the security definitions for EIPLBE as a generalization of the PLBE q-query security [26]. Also, see Remark 1.

Definition 4 (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 probability is at most $1/2 + \mathsf{negl}(\lambda)$:*

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

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 (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 probability is at most $1/2 + \mathsf{negl}(\lambda)$:*

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

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 6 (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 probability is at most* $1/2 + \mathsf{negl}(\lambda)$*:*

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

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 7 (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 probability is at most* $1/2 + \mathsf{negl}(\lambda)$*:*

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

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 8 (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 probability is at most* $1/2 + \mathsf{negl}(\lambda)$*:*

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

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 $a \neq b$.*

4.2 Building Indexed EITT from EIPLBE

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}.[12] 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 $(b, p, q) \leftarrow \mathsf{Index\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, i)$.

[12] 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).

    - 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.

```
Algorithm Index-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 $(0, \perp, \perp)$.
```

Fig. 2. Index-Trace

```
Algorithm ID-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 $\dfrac{p+q}{2} > \dfrac{\mathsf{count}_\ell}{N}$, set $\mathsf{id}_\ell = 0$. Else set $\mathsf{id}_\ell = 1$.

Finally, output $\mathsf{id}$.
```

Fig. 3. Index-Trace

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

Efficiency. If the scheme $\textsf{EIPLBE} = (\textsf{EIPLBE.Setup}, \textsf{EIPLBE.KeyGen}, \textsf{EIPLBE.Enc}, \textsf{EIPLBE.SplEnc}, \textsf{EIPLBE.Dec})$ is a EIPLBE scheme with $(\textsf{T-s}, \textsf{T-e}, \textsf{T-}\widetilde{\textsf{e}}, \textsf{T-k}, \textsf{T-d}, \textsf{S-c}, \textsf{S-k})$-efficiency, then the scheme $\textsf{TT} = (\textsf{Setup}, \textsf{KeyGen}, \textsf{Enc}, \textsf{Dec}, \textsf{Trace})$ is a (indexed keygen, public/private)-embedded identity tracing scheme with $(\textsf{T-s}', \textsf{T-e}', \textsf{T-k}', \textsf{T-d}', \textsf{T-t}', \textsf{S-c}', \textsf{S-k}')$-efficiency, where the efficiency measures are related as follows:

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

Security The security proof is included in the full version of our paper

References

1. Abdalla, M., Dent, A.W., Malone-Lee, J., Neven, G., Phan, D.H., Smart, N.P.: 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. pp. 361–376 (2007), `https://doi.org/10.1007/978-3-540-71677-8_24`
2. Agrawal, S., Bhattacherjee, S., Phan, D.H., Stehlé, D., Yamada, S.: 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. pp. 2277–2293 (2017), `http://doi.acm.org/10.1145/3133956.3134041`
3. Baltico, C.E.Z., Catalano, D., Fiore, D., Gay, R.: Practical functional encryption for quadratic functions with applications to predicate encryption. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. pp. 67–98 (2017)
4. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: CRYPTO. pp. 1–18 (2001)
5. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. J. ACM 59(2), 6 (2012)
6. Billet, O., Phan, D.H.: Efficient traitor tracing from collusion secure codes. In: Information Theoretic Security, Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings. pp. 171–182 (2008), `https://doi.org/10.1007/978-3-540-85093-9_17`

7. Böhl, F., Hofheinz, D., Jager, T., Koch, J., Seo, J.H., Striecks, C.: Practical signatures from standard assumptions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 461–485. Springer (2013)
8. Boneh, D., Franklin, M.K.: 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. pp. 338–353 (1999), https://doi.org/10.1007/3-540-48405-1_22
9. Boneh, D., Sahai, A., Waters, B.: Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: EUROCRYPT. pp. 573–592 (2006)
10. Boneh, D., Waters, B.: 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. pp. 211–220 (2006)
11. Boneh, D., Zhandry, M.: 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. pp. 480–499 (2014), https://doi.org/10.1007/978-3-662-44371-2_27
12. Chabanne, H., Phan, D.H., Pointcheval, D.: 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. pp. 542–558 (2005), https://doi.org/10.1007/11426639_32
13. Chen, Y., Vaikuntanathan, V., Waters, B., Wee, H., Wichs, D.: Traitor-tracing from lwe made simple and attribute-based. In: TCC (2018)
14. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: CRYPTO. pp. 257–270 (1994)
15. Chor, B., Fiat, A., Naor, M., Pinkas, B.: Tracing traitors. IEEE Trans. Information Theory 46(3), 893–910 (2000), https://doi.org/10.1109/18.841169
16. Coron, J., Lepoint, T., Tibouchi, M.: 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. pp. 476–493 (2013)
17. Coron, J., Lepoint, T., Tibouchi, M.: 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)
18. Döttling, N., Schröder, D.: Efficient pseudorandom functions via on-the-fly adaptation. In: Annual Cryptology Conference. pp. 329–350. Springer (2015)
19. Fazio, N., Nicolosi, A., Phan, D.H.: Traitor tracing with optimal transmission rate. In: Information Security, 10th International Conference, ISC 2007, Valparaíso, Chile, October 9-12, 2007, Proceedings. pp. 71–88 (2007), https://doi.org/10.1007/978-3-540-75496-1_5
20. Freeman, D.M.: 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. pp. 44–61. EUROCRYPT'10, Springer-Verlag, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-13190-5_3
21. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: EUROCRYPT (2013)
22. Garg, S., Kumarasubramanian, A., Sahai, A., Waters, B.: Building efficient fully collusion-resilient traitor tracing and revocation schemes. In: Proceedings of the

17th ACM Conference on Computer and Communications Security. pp. 121–130. CCS '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1866307.1866322

23. Gentry, C., Gorbunov, S., Halevi, S.: Graph-induced multilinear maps from lattices. In: TCC (2015)
24. Goyal, R., Koppula, V., Russell, A., Waters, B.: Risky traitor tracing and new differential privacy negative results. In: Crypto (2018)
25. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. In: 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017. pp. 612–621 (2017)
26. Goyal, R., Koppula, V., Waters, B.: Collusion resistant traitor tracing from learning with errors. In: STOC (2018)
27. Kiayias, A., Yung, M.: 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. pp. 450–465 (2002), https://doi.org/10.1007/3-540-46035-7_30
28. Kurosawa, K., Desmedt, Y.: 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. pp. 145–157 (1998), https://doi.org/10.1007/BFb0054123
29. Kurosawa, K., Yoshida, T.: 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. pp. 172–187 (2002), https://doi.org/10.1007/3-540-45664-3_12
30. Ling, S., Phan, D.H., Stehlé, D., Steinfeld, R.: 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. pp. 315–334 (2014), https://doi.org/10.1007/978-3-662-44371-2_18
31. Nishimaki, R., Wichs, D., Zhandry, M.: 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. pp. 388–419 (2016), https://doi.org/10.1007/978-3-662-49896-5_14
32. Phan, D.H., Safavi-Naini, R., Tonien, D.: 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. pp. 264–275 (2006), https://doi.org/10.1007/11787006_23
33. Staddon, J., Stinson, D.R., Wei, R.: Combinatorial properties of frameproof and traceability codes. IEEE Trans. Information Theory 47(3), 1042–1049 (2001), https://doi.org/10.1109/18.915661
34. Stinson, D.R., Wei, R.: Combinatorial properties and constructions of traceability schemes and frameproof codes. SIAM J. Discrete Math. 11(1), 41–53 (1998), https://doi.org/10.1137/S0895480196304246
35. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under LWE. In: 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017. pp. 600–611 (2017)