Anonymous Traitor Tracing: How to Embed Arbitrary Information in a Key

Ryo Nishimaki$^{1(\text{✉})}$, Daniel Wichs^{2}, and Mark Zhandry^{3}

^{1} NTT Secure Platform Laboratories, Tokyo, Japan
nishimaki.ryo@lab.ntt.co.jp
^{2} Northeastern University, Boston, USA
wichs@ccs.neu.edu
^{3} MIT/Princeton University, Cambridge, USA
mzhandry@princeton.edu

Abstract. In a traitor tracing scheme, each user is given a different decryption key. A content distributor can encrypt digital content using a public encryption key and each user in the system can decrypt it using her decryption key. Even if a coalition of users combines their decryption keys and constructs some "pirate decoder" that is capable of decrypting the content, there is a public tracing algorithm that is guaranteed to recover the identity of at least one of the users in the coalition given black-box access to such decoder.

In prior solutions, the users are indexed by numbers $1, \ldots, N$ and the tracing algorithm recovers the index i of a user in a coalition. Such solutions implicitly require the content distributor to keep a record that associates each index i with the actual identifying information for the corresponding user (e.g., name, address, etc.) in order to ensure accountability. In this work, we construct traitor tracing schemes where all of the identifying information about the user can be embedded directly into the user's key and recovered by the tracing algorithm. In particular, the content distributor does not need to separately store any records about the users of the system, and honest users can even remain anonymous to the content distributor.

The main technical difficulty comes in designing tracing algorithms that can handle an exponentially large universe of possible identities, rather than just a polynomial set of indices $i \in [N]$. We solve this by abstracting out an interesting algorithmic problem that has surprising connections with seemingly unrelated areas in cryptography. We also extend our solution to a full "broadcast-trace-and-revoke" scheme in which the traced users can subsequently be revoked from the system. Depending on parameters, some of our schemes can be based only on the existence of public-key encryption while others rely on indistinguishability obfuscation.

R. Nishimaki—This work was done while the author was visiting Northeastern University.

D. Wichs—Research supported by NSF grants CNS-1347350, CNS-1314722, CNS-1413964. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467.

M. Fischlin and J.-S. Coron (Eds.): EUROCRYPT 2016, Part II, LNCS 9666, pp. 388–419, 2016.
DOI: 10.1007/978-3-662-49896-5_14

1 Introduction

The Traitor-Tracing Problem. Traitor-tracing systems, introduced by Chor et al. [12], are designed to help content distributors identify the origin of pirate decryption boxes (such as pirate cable-TV set-top decoders) or pirate decryption software posted on the Internet.

In the traditional problem description, there is a set of legitimate users with numeric identities $[N] = \{1, \dots, N\}$ for some (large) polynomial N. Each user $i \in [N]$ is given a different decryption key sk_i. A content distributor can encrypt content under the public key pk of the system and each legitimate user i can decrypt the content with her decryption key sk_i. For example this could model a cable-TV network broadcasting encrypted digital content, where each legitimate customer i is given a set-top decoder with the corresponding decryption key sk_i embedded within it.

One of the main worries in this scenario is that a user might make copies of her key to re-sell or even post in a public forum, therefore allowing illegitimate parties to decrypt the digital content. While this cannot be prevented, it can be deterred by ensuring that such "traitors" are held accountable if caught. To evade accountability, a traitor might modify her secret key before releasing it in the hope that the modified key cannot be linked to her. More generally, a coalition of several traitors might come together and pool the knowledge of all of their secret keys to come up with some "pirate decoder" program capable of decrypting the digital content. Such a program could be made arbitrarily complex and possibly even obfuscated in the hopes that it will be difficult to link it to any individual traitor. A traitor-tracing scheme ensures that no such strategy can succeed – there is an efficient *tracing algorithm* which is given black-box access to any such pirate decoder and is guaranteed to output the numeric identity $i \in [N]$ of at least one of the traitors in the coalition that created the program.

Who Keeps Track of User Info? The traditional problem definition for traitor tracing makes an implicit assumption that there is an external mechanism to keep track of the users in the system and their identifying information in order to ensure accountability. In particular, either the content distributor or some third party would need to keep a record that associates the numeric identities $i \in [N]$ of the users with the actual identifying information (e.g., name, address, etc.). This way, if the tracing algorithm identifies a user with numeric identity i as a traitor, we can link this to an actual person.

Goal: Embedding Information in Keys. The main goal of our work is to create a traitor tracing system where all information about each user is embedded directly into their secret key and there is no need to keep any external record about the honest users of the system. More concretely, this goal translates to having a traitor tracing scheme with a *flexible*, exponential-size universe of identities $\mathcal{ID}$[1]. A user's identity $\mathsf{id} \in \mathcal{ID}$ can then be a string containing all relevant identifying information about the user. The content distributor has a *master secret key* msk, and for any user with identity $\mathsf{id} \in \mathcal{ID}$ the content provider can use msk to create a user secret key $\mathsf{sk}_{\mathsf{id}}$ with this information embedded inside it. The content provider does not need to keep any records about the user after the secret key is given out. If a coalition of traitors gets together and constructs a pirate decoder, the tracing algorithm should recover the entire identity id of a traitor involved in the coalition, which contains all of the information necessary to hold the traitor accountable.

Moreover, if we have such a traitor tracing scheme with an exponentially large universe of identities as described above, it is also possible to construct a fully *anonymous* traitor tracing system where the content provider never learns who the honest users are. Instead of a user requesting a secret key for identity $\mathsf{id} \in \mathcal{ID}$ by sending id to the content provider directly, the user and the content provider run a *multiparty computation* (MPC) where the user's input consists of the string id containing all of her identifying information (signed by some external identity verification authority), the content provider's input is msk, and the computation gives the user $\mathsf{sk}_{\mathsf{id}}$ as an output (provided that the signature verifies) and the content provider learns nothing. This can even be combined with an anonymous payment system such as bit-coin to allow users to anonymously pay for digital content. Surprisingly, this shows that anonymity and traitor tracing are not contradictory goals; we can guarantee anonymity for honest users who keep their decryption keys secret while still maintaining the ability to trace the identities of traitors.

Unfortunately, it turns out that prior approaches to the traitor tracing problem cannot handle large identities and crucially rely on the fact that, in the traditional problem definition, the set of identities $[N]$ is polynomial in size. We first survey the prior work on traitor tracing and then present our new results and techniques that allow us to achieve the above goals.

1.1 Prior Work

Traitor Tracing Overview. Traitor tracing was introduced by Chor et al. [12]. There are many variants of the problem depending on whether the encryption and/or the tracing algorithm are public key or secret key procedures, whether the tracing algorithm is black-box, and whether the schemes are "fully collusion resistant" (no bound on the number of colluding traitors), or whether they are "bounded collusion resistant". See e.g., the works of [6–9,11,13,17,19,29,31–34,37,38] and references within for a detailed overview of prior work.

In this work, we will focus on schemes with a public-key encryption and a public-key and black-box tracing algorithm, and will consider both fully and

[1] While schemes with exponential identity spaces are normally referred to as "identity-based", identity-based traitor tracing already has a defined meaning [1]. In particular, the space of identities that are traced in an identity-based traitor tracing scheme is still polynomial. We use the term "flexible" traitor tracing to refer to schemes where the space of identities that can be traced is exponential.

bounded collusion resistance. In all prior systems, the set of legitimate users was fixed to $[N] = \{1, \ldots, N\}$ for some large polynomial N, and the main differences between the prior schemes depends on how various parameters (public key size, secret key size, ciphertext size) scale with the number of users N.

Traitor Tracing via Private Broadcast Encryption (PLBE). Boneh et al. [7] build the first fully collusion resistant traitor tracing scheme where the ciphertext size is $O(\sqrt{N})$, private key size is $O(1)$, public key size is $O(\sqrt{N})$ (we ignore factors that are polynomial in the security parameter but independent of N). The scheme is based on bilinear groups. This work also presents a general approach for building traitor tracing schemes, using an intermediate primitive called *private linear broadcast encryption* (PLBE). We follow the same approach in this work and therefore we elaborate on it now.

A PLBE scheme can be used to create a ciphertext that can only be decrypted by users $i \in [N]$ with $i \leq T$ for some threshold value $T \in \{0, \ldots, N\}$ specified during encryption. Furthermore, the only way to distinguish between a ciphertext created with the threshold value T vs. T' for some $T < T'$ is to have a secret key sk_i with $i \in \{T, \ldots T' - 1\}$ that can decrypt in one case but not the other.

A PLBE scheme can immediately be used as a traitor-tracing scheme. The encryption algorithm of the tracing scheme creates a ciphertext with the threshold $T = N$, meaning that all users can decrypt it correctly. The tracing algorithm gets black-box access to a pirate decoder and does the following: it tries all thresholds $T = 1, \ldots, N$ and tests the decoder on ciphertext created with threshold T until it finds the first such threshold for which there is a "big jump" in the decryption success probability between T and $T - 1$. It outputs the index T as the identity of the traced traitor. The correctness of the above approach can be analyzed as follows. We know that the decoder's success probability on $T = 0$ is negligible (since such ciphertexts cannot be decrypted even given all the keys) and on $T = N$ it is large (by the correctness of the pirate decoder program). Therefore, there must be some threshold T on which there is a big jump in the success probability, but by the privacy property of the PLBE, a big jump can only occur if the secret key sk_T was used in the construction of the pirate decoder. Note that the run-time of this tracing algorithm is $O(N)$.

State of the Art Traitor Tracing via Obfuscation. Recently, Garg et al. [21] and Boneh and Zhandry [9] construct new fully collusion resistant traitor tracing scheme with essentially optimal parameters where key/ciphertext sizes only depend logarithmically on N. The schemes are constructed using the same PLBE framework as in [7] and the main contributions are the construction of a new PLBE scheme with the above parameters. These constructions both rely on indistinguishability obfuscation. More recently, Garg et al. [22] construct a PLBE with polylogarithmic parameters based on simple assumptions on multilinear maps. We note that in all three schemes, the PLBE can be extended to handle flexible (exponential) identity spaces by setting $N = 2^n$ for polynomial n. In this case, encryption and key generation, as well as ciphertext and secret key sizes, will grow polynomially in n. However, a flexible PLBE scheme does not directly yield to a flexible traitor tracing scheme. In particular, the tracing algorithm of [7] cannot be applied in this setting because it will run in exponential time, namely $O(2^n)$.

Broadcast Encryption, Trace and Revoke. We also mention work on a related problem called broadcast encryption. Similar to traitor tracing, such schemes have a collection of users $[N]$. A sender can create a ciphertext that can be decrypted by all of the users of the system *except* for specified set of "revoked users" (which may be colluding). See e.g., [16–18,20,24,26,32,34,39] and references within.

A trace and revoke system is a combination of broadcast encryption and traitor tracing [32,34]. In other words, once traitors are identified by the tracing algorithm they can also be revoked from decrypting future ciphertexts. Boneh and Waters [8] proposed a fully collusion resistant trace and revoke scheme where the private/public keys and ciphertexts are all of size $O(\sqrt{N})$. It was previously unknown how to obtain fully collusion resistant trace and revoke schemes with logarithmic parameter sizes. Separately, though, it is known how to build both broadcast encryption and traitor tracing with such parameters using obfuscation [9,21,41], and one could reasonably expect that it is possible to combine the techniques to obtain a broadcast, trace, and revoke system.

Watermarking. Lastly, we mention related work on watermarking cryptographic functions [14,15,35]. These works show how to embed arbitrary data into the secret key of a cryptographic function (e.g., a PRF) in such a way that it is impossible to create any program that evaluates the function (even approximately) but in which the mark is removed. This is conceptually related to our goal of embedding arbitrary data into the secret keys of users in a traitor-tracing scheme. Indeed, one could think of constructing a traitor tracing scheme where we take a standard public-key encryption scheme and give each user a watermarked version of the decryption key containing the user's identity embedded. Unfortunately, this solution does not work with current definitions of watermarking security, where we assume that each key can only be marked once with one piece of embedded data. In the traitor tracing scenario, we would want mark the same key many times with different data for each user. Conversely, solutions to the traitor tracing problem do not yield watermarking schemes since they only require us to embed data in carefully selected secret keys chosen by the scheme designer rather than in arbitrary secret keys chosen by the user.

1.2 Our Results

Our main result is to give new constructions of traitor-tracing schemes that supports a flexibly large space of identities $\mathcal{ID} = [2^n]$ where the parameter n is an arbitrary polynomial corresponding to the bit-length of the string $\mathsf{id} \in \mathcal{ID}$ which should be sufficiently large encode all relevant identifying information about the user. The user's secret key $\mathsf{sk}_{\mathsf{id}}$ contains the identity id embedded within it, so there is no need to keep any external record of users. The tracing algorithm recovers all of the identifying information id about a traitor directly from the pirate decoder. We construct such a scheme where the secret key $\mathsf{sk}_{\mathsf{id}}$ is of length $\mathsf{poly}(n)$, which is essentially optimal since it must contain the data id embedded within it. The first scheme we construct also has ciphertexts of size $\mathsf{poly}(n)$ but we then show how to improve this to ciphertexts of constant size independent of n (though still dependent on the security parameter). In the latter scheme, the identity length n need not be specified ahead of time: different users can potentially have different amounts of identifying information included in their key, and there is no restriction on the amount of information that can be included. The schemes are secure against an unbounded number of collusions.

Our schemes are secure assuming the existence of certain types of *private broadcast encryption*, which themselves are special cases of functional encryption (FE). Our work mainly focuses on building traitor tracing from these private broadcast schemes. We then instantiate the private broadcast schemes using recent constructions of FE, which in turn are built from indistinguishability obfuscation (iO) and one-way functions (OWF). An interesting direction for future work is to build private broadcast encryption from milder assumptions such as LWE.

We also construct schemes which are only secure against collusions of size at most q, where the ciphertext size is either of length $O(n)\mathsf{poly}(q)$ assuming only public-key encryption, or of only length $\mathsf{poly}(q)$ independent of n assuming sub-exponential LWE.[2] We also extend the above construction to a full trace and revoke scheme, allowing the content distributor to specify a set of revoked users during encryption. Assuming iO, we get such a scheme where neither the ciphertexts nor the secret keys grow with the set of revoked users.

1.3 Our Techniques

Our high level approach follows that of Boneh et al. [7], using PLBE as an intermediate primitive to construct traitor tracing. There are two main challenges: the first is to construct a PLBE scheme that supports an exponentially large identity space $\mathcal{ID} = [2^n]$ for some arbitrary polynomial n. The second, more interesting challenge, and the main focus of this work, is to construct a tracing algorithm which runs in time polynomial in n rather than $N = 2^n$.

PLBE with Large Identity Space. The work of Boneh and Zhandry [9] already constructs a PLBE scheme where the key/ciphertext size is polynomial in n. Unfortunately, the proof of security relies on a reduction that runs in time polynomial in $N = 2^n$ which is exponential in the security parameter. Thus going through their construction we would need to assume the sub-exponential hardness of iO (and OWFs) to get a secure PLBE. We instead take a different approach, suggested by [21], and construct PLBE directly from (indistinguishability based) functional encryption (FE). For technical reasons detailed below, we

[2] The above parameters ignore fixed polynomial factors in the security parameters.

actually need an adaptively secure PLBE scheme, and thus an adaptively secure FE scheme. In the unbounded collusion setting, these can be constructed from iO [2,40] or from simple assumptions on multilinear maps [22]. Alternatively, we get a PLBE scheme which is (adaptively) secure against a *bounded* number of collusions by relying on bounded-collusion FE which can be constructed from any public-key encryption [25] or from sub-exponential LWE if we want succinct ciphertexts [23].

A New Tracing Algorithm and the Oracle Jump-Finding Problem. The more interesting difficulty comes in making the tracing algorithm run in time polynomial in n rather than $N = 2^n$. We can think of the pirate decoder as an oracle that can be tested on PLBE ciphertexts created with various thresholds $T \in \{0, \dots, N\}$ and for any such threshold T it manages to decrypt correctly with probability p_T. For simplicity, let us think of this as an oracle that on input T outputs the probability p_T directly (since we approximate this value by testing the decoder on many ciphertexts). We know that p_0 is close to 0 and that p_N is the probability that a pirate decoder decrypts correctly, which is large – let's say $p_N = 1$ for simplicity. Moreover, we know that for any T, T' with $T < T'$ the values p_T and $p_{T'}$ are negligibly close *unless* there is a traitor with identity $i \in \{T, \dots T' - 1\}$, since encryptions with thresholds T and T' are indistinguishable. In particular this means that for any point T at which there is a "jump" so that $|p_T - p_{T-1}|$ is noticeable, corresponds to a traitor. Since we know that the number of traitors in the coalition is bounded by some polynomial, denoted by q, we know that there are at most q jumps in total and that there must be at least one "large jump" with a gap of at least $1/q$. The goal is to find at least one jump. We call this the "oracle jump-finding problem".

An Algorithm for the Oracle Jump-Finding Problem. The tracing algorithm of [7] essentially corresponds to a linear search and tests the oracle on every point $T \in [N]$ and thus takes at least $O(N)$ steps in the worst case to find a jump. When using flexibly large identity universes (that is, taking N to be exponential), the tracing algorithm will therefore run in exponential time. This is true *even if the underlying PLBE is efficient for such identity spaces*, including the PLBEs discussed above. Our goal is to design a better algorithm that takes at most $\mathsf{poly}(n, q)$ steps.

It is tempting to simply substitute binary search in place of linear search. We would first call the oracle on the point $T/2$ and learn $p_{T/2}$. Depending on whether the answer is closer to 0 or 1 we recursively search either the left interval or the right interval. The good news is in each step the size of the interval decreases by half and therefore there would be at most n steps. The bad news is that the gap in probabilities between the left and right end points now also decreases by a half and therefore after i steps we would only be guaranteed that the interval contains a jump with a gap of $2^{-i}/q$ which quickly becomes negligible.

Interestingly, we notice that the same oracle jump-finding problem implicitly appeared in a completely unrelated context in a work of Boyle et al. [10] showing the equivalence of indistinguishability obfuscation and a special case of differing-inputs obfuscation. Using the clever approach developed in the context of that work, we show how to get a $\mathsf{poly}(n, q)$ algorithm for the oracle jump finding problem and therefore an efficient tracing algorithm.

The main idea is to follow the same approach as binary search, but each time that the probability at the mid-point is noticeably far from both end-points we recurse on both the left and the right interval. This guarantees that there is always a large jump with a gap of at least $1/q$ within the intervals being searched. Furthermore, since the number of jumps is at most q we can bound the number of recursive steps in which both intervals need to be searched by q, and therefore guarantee that the algorithm runs in $\mathsf{poly}(n, q)$ steps.

Interestingly, due to our tracing algorithm choosing which T to test based on the results of previous tests, we need our PLBE scheme to be *adaptively* secure, and hence also the underlying FE scheme must be adaptively secure. This was not an issue in [7] for two reasons: (1) their tracing algorithm visits *all* $T \in [N]$, and (2) for polynomial N statically secure and adaptive secure PLBE are equivalent. Fortunately, as explained above, we know how to construct PLBE that is adaptively secure against unbounded collusions from iO or simple multilinear map assumptions. For the bounded collusion setting, we can obtain adaptively secure PLBE from public key encryption following [25].

We note that in an independent work, Kiayias and Tang [28] give another method of tracing in large identity spaces; however their analysis applies only to *random* user identities, and requires a means to verify that the identity outputted by the tracing algorithm actually corresponds to a one of the generated decryption keys. Our tracing algorithm does not have these limitations.

Tracing More General Decoders. In [7], a pirate decoder is considered "useful" if it decrypts the encryption of a random message with non-negligible probability, and their tracing algorithm is shown to work for such decoders. However, restricting to decoders that work for random messages is unsatisfying, as we would like to trace, say, decoders that work for very particular messages such as cable-TV broadcasts. The analysis of [7] appears insufficient for this setting. Kiayias and Yung [30] consider more general decoders, but their definition inherently places a lower bound on the min-entropy of the plaintext distribution. In our analysis, we show that even if a decoder can distinguish between two particular messages (of the adversary's choice) with non-negligible advantage, then it can be traced. To our knowledge, ours is the first traitor tracing system that can trace such general decoders.

Short Ciphertexts. In the above approach we construct traitor-tracing via a PLBE scheme where the ciphertext is encrypted with respect to some threshold $T \in \{0, \ldots, N\}$. The ciphertext must encode the entire information about T and is therefore of size at least $n = \log N$, which corresponds to the bit-length of the user's identifying information id. In some cases, if the size of id is truly large (e.g., the identifying information might contain a JPEG image of the user) we would want the ciphertext size to be much smaller than n. One trivial option is to first hash the user's identifying information, and use our tracing scheme above on the hashes. However, the tracer would then only learn the hash of the identifying information, and would need to keep track of the information and hashes to actually accuse a user. This prevents the scheme from being used in the anonymous setting.

Instead, we show how to have the tracer learn identifying information in its entirety by generalizing the PLBE approach in a way that lets us divide the user's identity into small blocks. Very roughly, we then trace the value contained in each block one at a time. The ciphertext now only needs to encode the block number that is currently being traced, and a single threshold for that block. This lets us reduce the ciphertext to size to only be proportional to $\log n$ rather than n. To do so we need to generalize the notion of PLBE which also leads to a generalization of the oracle-jump-finding problem and the algorithm that solves it. We note that since we can assume $n < 2^\lambda$, factors logarithmic in n can be absorbed into terms involving the security parameter. Thus our ciphertext size can actually be taken to be independent of the bit length of identities.

We implement our PLBE generalization using FE. As above, we need adaptive security, which corresponds to an adaptively secure FE scheme. We now also need the FE to have *compact* ciphertexts, whose size is independent of the functions being evaluated. In the unbounded collusion setting, a recent construction of Ananth and Sahai [4] shows how to build such an FE from iO. Moreover, in their FE scheme, the function size need not be specified a priori nor known during encryption time, and different secret keys can correspond to functions of different sizes. In our traitor tracing scheme, this translates to there being no a priori bound on the length of identities, and different users can have different amounts of identifying information embedded in their secret keys.

In the bounded collusion setting, we can obtain such an FE from LWE using [23], though the scheme is only statically secure; we then use complexity leveraging to obtain an adaptively secure scheme from sub-exponential LWE.

Trace and Revoke. Finally, we extend our traitor tracing scheme to a trace and revoke system where users can be revoked. It turns out that this problem reduces to the problem of constructing "revocable functional encryption" where the encryption algorithm can specify some revoked users which will be unable to decrypt. The ciphertext size is independent of the size of the revoke list, but we assume that the revoke list is known to all parties. We construct such a scheme from indistinguishability obfuscation using the technique of somewhere statistically binding (SSB) hashing [27]. However, we omit the details about the trace and revoke system due to the limited space. See the full version of this paper [36].

1.4 Outline

In Sect. 2, we give some definitions and notations that we will use in our work. In Sect. 3, we define the oracle jump-finding problem, and show how to efficiently solve it. In Sects. 4 and 5, we use the solution of the jump-finding problem to give our new traitor tracing schemes.

2 Preliminaries

Throughout this work, we will use the notation $[N]$ to mean the positive integers from 1 to N: $[N] = \{1, \dots, N\}$. We will also use the notation $[M, N]$ to denote the integers form M to N, inclusive. We will use $(M, N]$ as shorthand for $[M+1, N]$. We will use $[M, N]_{\mathbb{R}}$ to denote the *real* numbers between M and N, inclusive.

Next, we will define several of the cryptographic primitives we will be discussing throughout this work. We start with the definition of traitor tracing that we will be achieving. Then, we will define the primitives we will use to construct traitor tracing. In all of our definitions, there is an implicit security parameter λ, and "polynomial time" and "negligible" are with respect to this security parameter.

2.1 Traitor Tracing with Flexible Identities

Here we define traitor tracing. Our definition is similar to that of Boneh, Sahai, and Waters [7], though ours is at least as strong, and perhaps stronger. In particular, our definition allows for tracing pirate decoders that can distinguish between encryptions of any two messages, whereas [7] only allows for tracing pirate decoders that can decrypt random messages. In Sect. 4, we discuss why the analysis in [7] appears insufficient for our more general setting, but nevertheless show that tracing is still possible.

Definition 1. *Let $\mathcal{ID}$ be some collection of identities, and $\mathcal{M}$ a message space. A flexible traitor tracing scheme for $\mathcal{M}, \mathcal{ID}$ is a tuple of polynomial time algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ *where:*

- $\mathsf{Setup}()$ *is a randomized procedure with no input (except the security parameter) that outputs a master secret key* msk *and a master public key* mpk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ *takes as input the master secret* msk *and an identity* $\mathsf{id} \in \mathcal{ID}$, *and outputs a secret key* $\mathsf{sk}_{\mathsf{id}}$ *for* id.
- $\mathsf{Enc}(\mathsf{mpk}, m)$ *takes as input the master public key* mpk *and a message* $m \in \mathcal{M}$, *and outputs a ciphertext* c.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$ *takes as input the secret key* $\mathsf{sk}_{\mathsf{id}}$ *for an identity* id *and a ciphertext* c, *and outputs a message* m.
- $\mathsf{Trace}^{\mathcal{D}}(\mathsf{mpk}, m_0, m_1, q, \epsilon)$ *takes as input the master public key* mpk, *two messages* m_0, m_1, *and parameters* q, ϵ, *and has oracle access to a decoder algorithm* $\mathcal{D}$. *It produces a (possibly empty) list of identities* $\mathcal{L}$.
- **Correctness.** *For any message* $m \in \mathcal{M}$ *and identity* $\mathsf{id} \in \mathcal{ID}$, *we have that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c) = m : \begin{array}{l} (\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(), \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}), \\ c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m) \end{array}\right] = 1$$

- **Semantic security.** *Informally, we ask that an adversary that does not hold any secret keys cannot learn the plaintext m. This is formalized by the following experiment between an adversary $\mathcal{A}$ and challenger:*
  - *The challenger runs* $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$*, and gives* mpk *to* $\mathcal{A}$*.*
  - $\mathcal{A}$ *makes a challenge query where it submits two messages* m_0^*, m_1^**. The challenger chooses a random bit b, and responds with the encryption of* m_b^**:* $c^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b^*)$*.*
  - $\mathcal{A}$ *produces a guess b' for b. The challenger outputs 1 if $b' = b$ and 0 otherwise.*

  We define the semantic security advantage of $\mathcal{A}$ as the absolute difference between $1/2$ and the probability the challenger outputs 1. The public key encryption scheme is semantically secure if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.
- **Traceability.** *Consider a subset of colluding users that pool their secret keys and produce a "pirate decoder" that can decrypt ciphertexts. Call a pirate decoder $\mathcal{D}$ "useful" for messages m_0, m_1 if $\mathcal{D}$ can distinguish encryptions of m_0 from m_1 with noticeable advantage. Then we require that such a decoder can be traced using* Trace *to one of the identities in the collusion. This is formalized using the following game between an adversary $\mathcal{A}$ and challenger, parameterized by a non-negligible function ϵ:*
  - *The challenger runs* $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$ *and gives* mpk *to* $\mathcal{A}$*.*
  - $\mathcal{A}$ *is allowed to make arbitrary keygen queries, where it sends an identity* $\mathsf{id} \in \mathcal{ID}$ *to the challenger, and the challenger responds with* $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$*. The challenger also records the identities queries in a list $\mathcal{L}$.*
  - $\mathcal{A}$ *then produces a pirate decoder $\mathcal{D}$, two messages m_0^*, m_1^*, and a non-negligible value ϵ. Let q be the number of keygen queries made (that is, $q = |\mathcal{L}|$). The challenger computes* $\mathcal{T} \leftarrow \mathsf{Trace}^{\mathcal{D}}(\mathsf{mpk}, m_0^*, m_1^*, q, \epsilon)$ *as the set of accused users. The challenger says that the adversary "wins" one of the following holds:*
    - * *$\mathcal{T}$ contains any identity outside of $\mathcal{L}$. That is, $\mathcal{T} \setminus \mathcal{L} \neq \emptyset$ or*
    - * *Both of the following hold:*
      - · *$\mathcal{D}$ is ϵ-useful, meaning* $\Pr[\mathcal{D}(c) = m_b^* : b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b^*)] \geq \frac{1}{2} + \epsilon$[3]*.*
      - · *$\mathcal{T}$ does not contain at least one user inside $\mathcal{L}$. That is, $\mathcal{T} \cap \mathcal{L} = \emptyset$.*

    The challenger then outputs 1 if the adversary wins, and zero otherwise.

[3] Checking the "winning" condition requires computing the probabilities a procedure outputs a particular value, which is in general an inefficient procedure. Thus our challenger as described is not an efficient challenger. However, it is possible to efficiently estimate these probabilities by running the procedure many times, and reporting the fraction of the time the particular value is produced. We could have instead defined our challenger to estimate probabilities instead of determine them exactly, in which case the challenger would be efficient. The resulting security definition would be equivalent.

We define the tracing advantage of $\mathcal{A}$ as the probability the challenger outputs 1. We say the public key encryption scheme is traceable if, for all PPT adversaries $\mathcal{A}$ and all non-negligible ϵ, the advantage of $\mathcal{A}$ is negligible.

2.2 Private Broadcast Encryption

In our traitor tracing constructions, it will be convenient for us to use a primitive we call *private broadcast encryption*, which is a generalization of the private *linear* broadcast encryption of Boneh et al. [7]. A private broadcast scheme is a broadcast scheme where the recipient set is hidden. Usually, the collection of possible recipient subsets is restricted: for example, in private linear broadcast encryption, the possible recipient sets are simply intervals. It will be useful for us to consider more general classes of recipient sets, especially for our short-ciphertext traitor tracing construction in Sect. 5.

Definition 2. *Let $\mathcal{ID}$ be the set of identities. Let $\mathcal{S}$ be a collection of subsets of $\mathcal{ID}$. Let $\mathcal{M}$ be a message space. A Private Broadcast Encryption (PBE) scheme is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $\mathsf{Setup}()$ *is a randomized procedure with no input (except the security parameter) that outputs a master secret key* msk *and a master public key* mpk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ *takes as input the master secret* msk *and a user identity* $\mathsf{id} \in \mathcal{ID}$. *It outputs a secret key* $\mathsf{sk}_{\mathsf{id}}$ *for* id.
- $\mathsf{Enc}(\mathsf{mpk}, S, m)$ *takes as input the master public key* mpk, *a* secret *set $S \in \mathcal{S}$, and a message $m \in \mathcal{M}$. It outputs a ciphertext c.*
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$ *takes as input the secret key* $\mathsf{sk}_{\mathsf{id}}$ *for a user* id, *and a ciphertext c. It outputs a message $m \in \mathcal{M}$ or a special symbol $\perp$.*
- ***Correctness.*** *For a secret set $S \in \mathcal{S}$, any identity* $\mathsf{id} \in S$, *any identity* $\mathsf{id}' \notin S$, *any message $m \in \mathcal{M}$, we have that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c) = m : \begin{array}{l}(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(), \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}), \\ c \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, m)\end{array}\right] = 1$$

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}'}, c) = \perp : \begin{array}{l}(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(), \mathsf{sk}_{\mathsf{id}'} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}'), \\ c \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, m)\end{array}\right] = 1$$

  In other words, a user id *is "allowed" to decrypt if* id *is in the secret set S. We also require that if* id *is not "allowed" (that is, if* $\mathsf{id} \notin S$*), then* Dec *outputs* $\perp$.
- ***Message and Set Hiding.*** *Intuitively, we ask that for* id *that are not explicitly allowed to decrypt a ciphertext c, that the message is hidden. We also ask that nothing is learned about the secret set S, except for what can be learned by attempting decryption with various* $\mathsf{sk}_{\mathsf{id}}$ *available to the adversary. These two requirements are formalized by the following experiment between an adversary $\mathcal{A}$ and challenger:*
  - *The challenger runs* $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$, *and gives* mpk *to $\mathcal{A}$.*

- *$\mathcal{A}$ is allowed to make arbitrary keygen queries, where it sends an identity* $\mathsf{id} \in \mathcal{ID}$ *to the challenger, and the challenger responds with* $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$*. The challenger also records* id *in a list $\mathcal{L}$.*
- *At some point, $\mathcal{A}$ makes a single challenge query, where it submits two secret sets $S_0^*, S_1^* \in \mathcal{S}$, and two messages m_0^*, m_1^*. The challenger flips a random bit $b \in \{0,1\}$, and computes the encryption of m_b^* relative to the secret set S_b^*: $c^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, S_b^*, m_b^*)$. Then, the challenger makes the following checks, which ensure that the adversary cannot trivially determine b from c^*:*
  * *If $m_0^* \neq m_1^*$, then successful decryption of the challenge ciphertext would allow determining b. Therefore, the challenger requires that none of the identities the adversary has the secret key for can decrypt the ciphertext. In other words, for any* $\mathsf{id} \in \mathcal{L}$, $\mathsf{id} \notin S_0^*$ *and* $\mathsf{id} \notin S_1^*$*. In other words, the sets $L \cap S_0^*$ and $L \cap S_1^*$ must be empty.*
  * *If $S_0^* \neq S_1^*$, then successful decryption for S_b^* but not for S_{1-b}^* would allow for determining b (even if $m_0^* = m_1^*$). Therefore, the challenger requires that all of the identities the adversary has secret keys for can either decrypt in both cases, or can decrypt in neither. In other words, for any* $\mathsf{id} \in L$, $\mathsf{id} \notin S_0^* \Delta S_1^*$*, where Δ denotes the symmetric difference operator. Notice that this check is redundant if $m_0^* \neq m_1^*$.*

  If either check fails, the challenger outputs a random bit and aborts the game. Otherwise, the challenger sends c^* to $\mathcal{A}$.
- *$\mathcal{A}$ is allowed to make additional keygen queries for arbitrary identities* id^**, subject to the constraint that* id *must satisfy the same checks as above: if $m_0^* \neq m_1^*$, then* $\mathsf{id} \notin S_0^*$ *and* $\mathsf{id} \notin S_1^*$*, and if $S_0^* \neq S_1^*$, then* $\mathsf{id} \notin S_0^* \Delta S_1^*$*. If the adversary tries to query in an* id *that fails the check, the challenger outputs a random bit and aborts the game.*
- *$\mathcal{A}$ outputs a guess b' for b. The challenger outputs 1 if $b' = b$ and 0 otherwise.*

We define the advantage of $\mathcal{A}$ as the absolute difference between $1/2$ *and the probability the challenger outputs 1. We say the private broadcast system is secure if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.*

For a private broadcast scheme, we call the collection $\mathcal{S}$ of secret sets the *secret class.* We are interested in several metrics for a private broadcast scheme:

- **Ciphertext size.** Notice that the ciphertext, while hiding the secret set S, information-theoretically contains enough information to reveal S: given the secret key for every identity, S can be determined by attempting decryption with every secret key. It must also contain enough information to entirely reconstruct the message m. Thus, we must have $|c| \geq \log|\mathcal{S}| + \log|\mathcal{M}|$. We will say the ciphertext size is *optimal* if $|c| \leq \mathsf{poly}(\lambda, \log|\mathcal{S}|) + \log|\mathcal{M}|$.
- **Secret key size.** Assuming the public and secret classes $\mathcal{P}, \mathcal{S}$ are expressive enough, from the secret key $\mathsf{sk}_{\mathsf{id}}$ for identity id, it is possible to reconstruct the entire identity id by attempting to decrypt ciphertexts meant for various subsets. Therefore, $|\mathsf{sk}_{\mathsf{id}}| \geq \log|\mathcal{ID}|$. We will say the user secret key size is *optimal* if $|\mathsf{sk}_{\mathsf{id}}| \leq \mathsf{poly}(\lambda, \log|\mathcal{ID}|)$.

- **Master key size.** The master public and secret keys do not necessarily encode any information, and therefore could be as short as $O(\lambda)$. We will say the master key sizes are *optimal* if $|\mathsf{msk}|, |\mathsf{mpk}| \leq \mathsf{poly}(\lambda)$.

Notice that in the case where $\mathcal{S} = \{\mathcal{ID}\}$, our notion of private broadcast reduces to the standard notion of (identity-based) broadcast encryption, and the notions of optimal ciphertext, user secret key, and master key sizes coincide with the standard notions for broadcast encryption.

2.3 Functional Encryption

Definition 3. *Let $\mathcal{M}$ be some message space, $\mathcal{Y}$ some other space, and $\mathcal{F}$ be a class of functions $f : \mathcal{M} \to \mathcal{Y}$. A Functional Encryption (FE) scheme for $\mathcal{M}, \mathcal{Y}, \mathcal{F}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $\mathsf{Setup}()$ *is a randomized procedure with no input (except the security parameter) that outputs a master secret key* msk *and a master public key* mpk*.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$ *takes as input the master secret* msk *and a function $f \in \mathcal{F}$. It outputs a secret key* sk_f *for f.*
- $\mathsf{Enc}(\mathsf{mpk}, m)$ *takes as input the master public key* mpk *and a message $m \in \mathcal{M}$, and outputs a ciphertext c.*
- $\mathsf{Dec}(\mathsf{sk}_f, c)$ *takes as input the secret key* sk_f *for a function $f \in \mathcal{F}$ and a ciphertext c, and outputs some $y \in \mathcal{Y}$, or $\perp$.*
- **Correctness.** *For any message $m \in \mathcal{M}$ and function $f \in \mathcal{F}$, we have that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, c) = f(m) : \begin{array}{l}(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(), \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f),\\ c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m)\end{array}\right] = 1$$

- **Security.** *Intuitively, we ask that the adversary, given secret keys $f_1, \ldots, f_n$, learns $f_i(m)$ for each i, but nothing else about m. This is formalized by the following experiment between an adversary $\mathcal{A}$ and challenger:*
  - *The challenger runs* $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$*, and gives* mpk *to $\mathcal{A}$.*
  - *$\mathcal{A}$ is allowed to make arbitrary keygen queries, where it sends a function $f \in \mathcal{F}$ to the challenger, and the challenger responds with* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$*. The challenger also records f in a list $\mathcal{L}$.*
  - *At some point, $\mathcal{A}$ makes a single challenge query, where it submits two messages m_0^*, m_1^*. The challenger checks that $f(m_0^*) = f(m_1^*)$ for all $f \in L$. If the check fails (that is, there is some $f \in L$ such that $f(m_0^*) \neq f(m_1^*)$), then the challenger outputs a random bit and aborts. Otherwise, the challenger flips a random bit $b \in \{0, 1\}$, and responds with the ciphertext $c^* \leftarrow$* $\mathsf{Enc}(\mathsf{mpk}, m_b^*)$*.*
  - *$\mathcal{A}$ is allowed to make additional keygen queries for functions $f \in \mathcal{F}$, subject to the constraint that $f(m_0^*) = f(m_1^*)$.*
  - *$\mathcal{A}$ outputs a guess b' for b. The challenger outputs 1 if $b' = b$ and 0 otherwise.*

  We define the advantage of $\mathcal{A}$ as the absolute difference between $1/2$ and the probability the challenger outputs 1. We say the functional encryption scheme is secure if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.

For a functional encryption scheme, we will be interested in the size of the various parameters (in addition to the security of the system itself):

– **Ciphertext size.** At a minimum, the ciphertext must information-theoretically encode the entire message (assuming the class $\mathcal{F}$ is expressive enough). Therefore $|c| \geq \log|\mathcal{M}|$. We will consider a scheme to have *optimal* ciphertext size if $|c| \leq \mathsf{poly}(\lambda, \log|\mathcal{M}|)$[4].
– **Secret key size.** The secret key must information-theoretically encode the entire function f, so $|\mathsf{sk}_f| \geq \log|\mathcal{F}|$. However, because we are interested in efficient algorithms, we cannot necessarily represent functions f using $\log|\mathcal{F}|$ bits, and may therefore need larger keys. Generally, f will be a circuit of a certain size, say s. We will say a scheme has *optimal* secret key size if $|\mathsf{sk}_f| \leq \mathsf{poly}(\lambda, s)$.
– **Master key size.** The master public and secret keys do not necessarily encode any information, and therefore could be as short as $O(\lambda)$. We will say the master key sizes are *optimal* if $|\mathsf{msk}|, |\mathsf{mpk}| \leq \mathsf{poly}(\lambda)$.

Construction. A construction of FE that has above properties is proposed by Ananth and Sahai [4]. The construction is based on indistinguishability obfuscation for circuits and one-way function.

3 An Oracle Problem

Here we define the oracle jump finding problem, which abstracts the algorithmic problem underlying both the iO/diO (differing-inputs obfuscation) conversion of [10] as well as the tracing algorithm in this work.

Definition 4. *The (N, q, δ, ϵ) jump finding problem is the following. An adversary chooses a set $C \subseteq [1, N]$ of q unknown points. Then, the adversary provides an oracle $P : [0, N] \to [0, 1]_{\mathbb{R}}$ such that:*

– *$|P(N) - P(0)| > \epsilon$. That is, over the entire domain, P varies significantly.*
– *For any $x, y \in [0, N], x < y$ in interval $(x, y]$ that does not contain any points in C (that is, $(x, y] \cap C = \emptyset$), it must be $|P(x) - P(y)| < \delta$. That is, outside the points in C, P varies very little.*

Our goal is to interact with the oracle P and output some *element in C.*

A pictorial representation of the jump finding problem is given in Fig. 1.

Notice that if $\epsilon < q\delta$, it is possible to have all adjacent values $P(x-1), P(x)$ be at less than δ apart, even for $x \in C$. Thus it becomes information-theoretically *im*possible to determine an $x \in C$. In contrast, for $\epsilon \geq q\delta$, if we query the oracle on all points there must exist some point x such that $|P(x) - P(x-1)| > \delta$, and this point must therefore belong to C. Therefore, this problem is inefficiently solvable $\epsilon \geq q\delta$. The following shows that for ϵ somewhat larger that $q\delta$, the problem can even be solved efficiently:

[4] This property has been referred to as “compactness” [3,5].

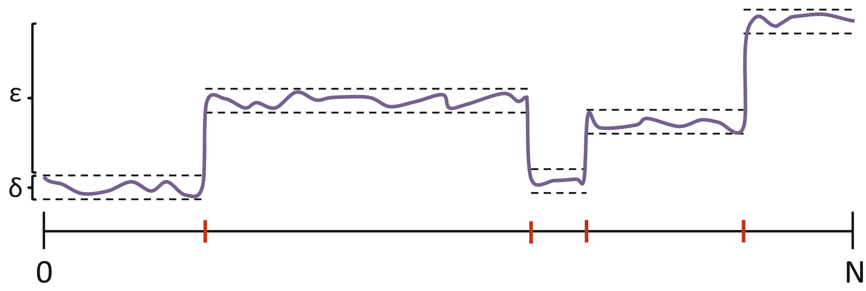

Fig. 1. Example of an oracle P when C contains 4 points. The purple curve represents the outputs of the oracle P on inputs in the interval $[0, N]$. The red hatch marks on the number line indicate the positions of the elements in C. The horizontal dashed lines show that, between the points in C, P is never changes more than δ. At the points in C, P can make arbitrary jumps in either direction.

Theorem 1. *There is a deterministic algorithm* $\mathsf{PTrace}^P(N, q, \delta)$ *that runs in time* $t = \mathsf{poly}(\log N, q)$ *(and in particular makes at most* t *queries to* P*) that will output at least one element in* C*, provided* $\epsilon \geq \delta(2+(\lceil \log N \rceil - 1)q)$*. Furthermore, the algorithm never outputs an element outside* C*, regardless of the relationship between* ϵ *and* δ*.*

Proof. We assume that $P(N) - P(0) > \epsilon$. The general case can be solved by running our algorithm once, and then running it a second time with the oracle $P'(x) = 1 - P(x)$, and outputting the union of the elements produced. We will also assume $N = 2^n$ is a power of 2, the generalization to arbitrary N being straightforward.

The starting point is the observation that if C contains only a single element x, then this problem is easily solved using binary search. Indeed, we can query P on $0, N/2, N$. If $x \in (0, N/2]$, then there are no points in C that are in $(N/2, N]$, and therefore $P(N) - P(N/2) < \delta$. This implies $P(N/2) - P(0) > \epsilon - \delta > \delta$. Similarly, if $x \in (N/2, N]$, then $P(N/2) - P(0) < \delta < \epsilon - \delta < P(N) - P(N/2)$. Therefore, it is easy to determine which half of $(0, N]$ x lies in. Moreover, on the half that x lies in, P still varies by $\epsilon' = \epsilon - \delta$. Therefore, we can recursively search for x on that half. Each time, we split the interval in which x lies in half, and decrease the total variation on that interval by only an additive δ. Since we perform at most $\log N$ steps in this binary search, the total variation will decrease by at most $\delta \log N$, and our choice of ϵ guarantees that the variation stays greater than δ. Therefore, we can proceed all the way down until we've isolated the point x, which we then output.

The problem arises when C contains more than just a single point. In this case, there may be points in both halves of the interval. If we recurse on both halves, the resulting algorithm will run in time that grows with N as opposed to $\log N$. The other option is to pick a single half-interval arbitrarily, and recurse only on that half. However, if there are points in C among both half-intervals, the variation in each half-interval may decrease by a factor of two. Recursing in this way will quickly cut the total variation down to below the threshold δ, at which point we will not be able to tell which intervals have points in C and which do not. Therefore, we need to be careful in how we choose which intervals to recurse on.

First we define a recursive algorithm $\mathsf{PTrace}_0^P(I, q, \delta)$ which takes as input an interval $I = (a, b]$, as well as q, δ. For any interval $I = (a, b]$, let $|I| = b - a$ be the number of points in I and let q_I be the number of points of C in I: $q_I = |I \cap C|$. Define $\Delta_I = P(b) - P(a)$. $\mathsf{PTrace}_0^P(I, q, \delta)$ works as follows:

- Let $I = (a, b]$. Query P on a, b to obtain $P(a), P(b)$. Compute $\Delta_I = P(b) - P(a)$
- If $\Delta_I \leq \delta$, abort and output the empty list $\mathcal{T} = \{\}$
- Otherwise, if $|I| = 1$, output $\mathcal{T} = \{b\}$
- Otherwise, partition I into two equal disjoint intervals I_L, I_R so that $I_L \cap I_R = \emptyset$, $I_L \cup I_R = I$, and $|I_L|, |I_R| = |I|/2$. Run $\mathcal{T}_L = \mathsf{PTrace}_0^P(I_L, q, \delta)$ and $\mathcal{T}_R = \mathsf{PTrace}_0^P(I_R, q, \delta)$. Output $\mathcal{T} = \mathcal{T}_L \cup \mathcal{T}_R$.

We then define PTrace to run PTrace_0 on the entire domain $(0, N]$: $\mathsf{PTrace}^P(N, q, \delta) = \mathsf{PTrace}_0^P((0, N], q, \delta)$. We now make several claims about PTrace_0. The first follows trivially from the definition of PTrace_0:

Claim. Any element outputted by PTrace_0 on interval I must be in $C \cap I$. In particular, any element outputted by PTrace is in C. Moreover, we have that any element s outputted must have $P(s) - P(s-1) > \delta$

Claim. The running time of PTrace is a polynomial in q and in $n = \log N$.

Proof. The running time of PTrace is dominated by the number of calls made to PTrace_0. We observe that the intervals I on which PTrace_0 is potentially called form a binary tree: the root is the entire interval $(0, N]$, the leaves are the singleton intervals $(x-1, x]$, and each non-leaf node corresponding to interval I has two children corresponding to intervals I_L and I_R that are the left and right halves of I. This tree has $1 + \log N$ levels, where the intervals in level i have size 2^i. Based on the definition of PTrace_0, PTrace_0 is only called on an interval I if I's parent contains at least one point in C, or equivalently that I or its sibling contain at least one point in C. Since there are only q points in C, PTrace is called on at most $2q$ intervals in each level. Thus the total number of calls, and hence the overall running time, is $O(q \log N)$.

Claim. Define $\alpha(I) \equiv \delta(\log|I| + (n-1)q_I - (n-2))$ where $n = \log N$. Any call to PTrace_0 with $q_I \geq 1$ and $\Delta_I > \alpha(I)$ will output *some* element.

Proof. If $|I| = 1$ and $q_I = 1$, then $\alpha(I) = \delta((n-1) - (n-2)) = \delta$. We already know that if $\Delta_I > \delta = \alpha(I)$, PTrace will output an element. Therefore, the claim holds in the case where $|I| = 1$.

Now assume the claim holds if $|I| \leq r$. We prove the case $|I| = r + 1$. Assume $q_I \geq 1$, and running PTrace_0 on I does not give any elements in C. Then running

PTrace_0 on I_L and I_R does not give any elements. For now, suppose q_{I_L} and q_{I_R} both positive. By induction this means that $\Delta_{I_L} \leq \alpha(I_L) = \delta(\log|I_L| + (n-1)q_{I_L} - (n-2))$ and $\Delta_{I_R} \leq \alpha(I_R) = \delta(\log|I_R| + (n-1)q_{I_R} - (n-2))$. Recall that $\log|I_R| = \log|I_L| = \log|I| - 1$. Together this means that $\Delta_I \leq \alpha(I_L)+\alpha(I_R) \leq \delta(\log|I|+(n-1)q_I-(n-2)-(n-\log|I|)) = \alpha(I)-(n-\log|I|)$. Since $\log|I| \leq n$, we have that $\Delta_I \leq \alpha(I)$.

Now suppose $q_{I_L} = 0$, which implies $q_{I_R} = q_I > 0$. The case $q_{I_R} = 0$ is handled similarly. Then $\Delta_{I_L} \leq \delta$, and by induction $\Delta_{I_R} \leq \alpha(I_R) = \delta(\log|I| + (n-1)q_I - (n-1))$. Thus $\Delta_I \leq \delta(\log|I| + (n-1)q_I - (n-1) + 1) = \alpha(I)$, as desired. This completes the proof of the claim. □

Notice that $\alpha(\ (0,N]\ ) = \delta(2 + (n-1)q) \leq \epsilon$. Also notice that by definition $\Delta_{(0,N]} > \epsilon$. Therefore, the initial call to PTrace_0 by PTrace outputs *some* element, and that element is necessarily in C. □

Now we define a related oracle problem, that takes the jump finding problem above, hides the oracle P inside a noisy oracle Q, and only provides us with the noisy oracle Q.

Definition 5. *The* (N, q, δ, ϵ) noisy *jump finding problem is as follows. An adversary chooses a set* $C \subseteq [1, N]$ *of* q *unknown points. The adversary then builds an oracle* $P : [0, N] \to [0, 1]_{\mathbb{R}}$ *as above, but does not provide it directly. As before,* P *must satisfy:*

- $|P(N) - P(0)| > \epsilon$
- *For any* $x, y \in [0, N], x < y$ *in interval* $(x, y]$ *that does not contain any points in* C *(that is,* $(x, y] \cap C = \emptyset$*), it must be* $|P(x) - P(y)| < \delta$.

Instead of interacting with P*, we interact with a randomized oracle* $Q : [0, N] \to \{0, 1\}$ *defined as follows:* $Q(x)$ *chooses and outputs a random bit that is 1 with probability* $P(x)$*, and 0 otherwise. A fresh sample is chosen for repeated calls to* $Q(x)$*, and is independent of all other samples outputted by* Q*. Our goal is to interact with the oracle* Q *and output* some *element in* C.

Theorem 2. *There is a probabilistic algorithm* $\mathsf{QTrace}^Q(N, q, \delta, \lambda)$ *that runs in time* $t = \mathsf{poly}(\log N, q, 1/\delta, \lambda)$ *(and in particular makes at most* t *queries to* O*) that will output at least one element in* C *with probability* $1 - \mathsf{negl}(\lambda)$*, provided* $\epsilon > \delta(5+2(\lceil\log N\rceil-1)q)$*. Furthermore, the algorithm never outputs an element outside* C*, regardless of the relationship between* ϵ *and* δ.

The idea is to, given Q, approximate the underlying oracle P, and run PTrace on the approximated oracle. Similar to the setting above, QTrace works even for "cheating" oracles P, as long as $|P(x) - P(y)| < \delta$ for all queried pairs x, y such that $(x, y]$ contains no points in C. We still need Q to be honestly constructed given P.

Proof. Our basic idea is to use O to simulate an approximation $\hat{P}$ to the oracle P, and then run PTrace using the oracle $\hat{P}$.

$\mathsf{QTrace}^Q(N, q, \delta, \epsilon, \lambda)$ works as follows. It simulates $\mathsf{PTrace}(N, q, \delta)$. Whenever PTrace queries P on input x, QTrace does the following:

– For $i = 1, \dots, O(\lambda/\delta^2)$, sample $z_i \leftarrow O(x)$
– Output $\hat{p}_x$ as the mean of the z_i.

Then QTrace outputs the output of PTrace.

As PTrace makes $O(q \log N)$ oracle calls to P, QTrace will make $O(\lambda q \log N/\delta^2)$ oracle calls. Moreover, the running time is bounded by this quantity as well. Therefore QTrace has the desired running time.

With probability at least $1-2^{-\lambda}$, we have that $|p_x - \hat{p}_x| < \delta/2$ for each x that are queried. This means that, with overwhelming probability, for all intervals $(x, y]$ that do not contain any elements of x, we have that $|p_y - p_x| < \delta$, so $|\hat{p}_y - \hat{p}_x| < 2\delta$ with overwhelming probability. Moreover, $|p_N - p_0| > \epsilon$, so $|\hat{p}_N - \hat{p_0}| > \epsilon - \delta$. Thus with overwhelming probability the oracle $\hat{P}$ seen by PTrace is an instance of the $(N, q, \delta' = 2\delta, \epsilon' = \epsilon - \delta)$ noiseless jump finding problem. Notice that

$$\epsilon' = \epsilon - \delta > \delta(5 + 2(n-1)q) - \delta = (2\delta)(2 + (n-1)q) = \delta'(2 + (n-1)q)$$

Therefore, $\hat{P}$ satisfies the conditions of Theorem 1, and PTrace outputs at least one element in C. QTrace outputs the same element, completing the proof.

Remark 1. We note that PTrace^P and QTrace^Q work even for "cheating" P that do not satisfy $|P(x) - P(y)| < \delta$ for *all* $(x, y]$ which do not intersect C, as long as the property holds for all pairs x, y that where queried by PTrace or QTrace^Q. This will be crucial for traitor tracing.

3.1 The Generalized Jump Finding Problem

Here we define a more general version of the jump finding problem that will be useful for obtaining short-ciphertext traitor tracing. In this version, the domain of the oracle P is an $r \times 2N$ grid that is short but wide (that is, $r \ll N$). The elements in C correspond to non-crossing curves between grid points from the top of the grid to the bottom, which divide the grid into $|C| + 1$ contiguous regions. The probabilities outputted by P are restricted to vary negligibly across each continuous region, but are allowed to vary arbitrary between different regions. The goal is to recover the complete description of *some* curve in C. To help make the problem tractable, we require that each curve is confined to oscillate about an *odd* column of the grid. Such curves can be represented by an integer $s \in [N]$ giving the position $2s-1$ of the column, and a bit string $b = (b_1, \dots, b_r) \in \{0,1\}^r$ specifying which side of the column the curve is on at each row. A pictorial representation of the generalized jump finding problem is given in Fig. 2, and a precise definition is given below.

Definition 6. *The $(N, r, q, \delta, \epsilon)$ generalized jump finding problem is the following. The adversary chooses a set C of q unknown tuples $(s, b_1, \dots, b_r) \in [N] \times \{0,1\}^r$ such that the s are distinct. Each tuple $(s, b_1, \dots, b_r)$ describes a curve between grid points from the top to bottom of the grid $[1, r] \times [0, 2N]$,* *which oscillates about the column at position $2s-1$, with $b=(b_1,\ldots,b_r)$ specifying which side of the column the curve is on at each row. These curves divide the grid into $|C|+1$ contiguous regions. For each pair $(i,x)\in[1,r]\times[0,2N]$ the adversary chooses a probability $p_{i,x}\in[0,1]_{\mathbb{R}}$ such that $p_{i,x}$ varies "minimally" within each contiguous region. We also require that overall from left to right, there is "significant" variation of the $p_{i,x}$. Formally, this means:*

- *For any pair of pairs of the form $(i,2x),(j,2x)\in[1,r]\times[0,2N]$, $|p_{i,2x}-p_{j,2x}|<\delta$. In other words, since curves in C are restricted to oscillate around* odd *columns, no curve crosses between points on the same* even *column, so each even column lies entirely in a single contiguous region. We therefore require that the probabilities associated with any two points on the same* even *column are close.*
- *Let C_i be the set of values $2s-b_i$ for tuples in C. C_i is then the set of grid points in the ith row that are immediately to the right of curves in C. For any two pairs $(i,x),(i,y)\in[1,r]\times[0,2N]$ in the same row such that the interval $(x,y]$ does not contain any points in C_i then $|p_{i,x}-p_{i,y}|<\delta$. In other words, if no curves cross between points in the same row, those points must be in the same contiguous region and therefore have close probabilities.*
- *We also make the requirement that the probabilities in the 0th column are identical, and the probabilities in the $2N$th column are identical. That is, $p_{i,0}=p_{i',0}$ for all $i,i'\in[r]$ and $p_{i,2N}=p_{i',2N}$ for all $i,i'\in[r]$. Define $p_0=p_{i,0}$ and $p_{2N}=p_{i,2N}$.*
- *Finally, $|p_{2N}-p_0|>\epsilon$. That is, the 0th and $2N$th columns have very different probabilities.*

We are now presented with one of two oracles, depending on the version of the problem:

- *In the noiseless version, we are given an oracle for the $p_{i,x}$: we are given oracle access to the function $P:[1,r]\times[0,2N]\to[0,1]_{\mathbb{R}}$ such that $P(i,x)=p_{i,x}$.*
- *In the noisy version, we are given a randomized oracle Q with domain $[1,r]\times[0,2N]$ that, on input (i,x), outputs 1 with probability $p_{i,x}$. Repeated calls to Q on the same x yield a fresh bit sampled independently.*

Our goal is to output some *element in C.*

Theorem 3. *There are algorithms $\mathsf{PTrace}'^{P}(N,r,q,\delta)$ and $\mathsf{QTrace}'^{Q}(N,r,q,\delta,\lambda)$ for the noiseless and noisy versions of the (N,r,q,δ,ϵ) generalized jump finding problem that run in time $\mathsf{poly}(\log N,r,q,1/\delta)$ and $\mathsf{poly}(\log N,r,q,1/\delta,\lambda)$, respectively, and output an element in C with overwhelming probability, provided $\epsilon>\delta(4+2(\lceil\log N\rceil-1)q)$ (for the noiseless case), or $\epsilon>\delta(9+4(\lceil\log N\rceil-1)q)$ (for the noisy case).*

This theorem is proved analogously to Theorems 1 and 2, and appears in below. Again, $\mathsf{PTrace}',\mathsf{QTrace}'$ work even if the oracle P is "cheating", as long as the requirements on P hold for all points queried by PTrace' or QTrace'.

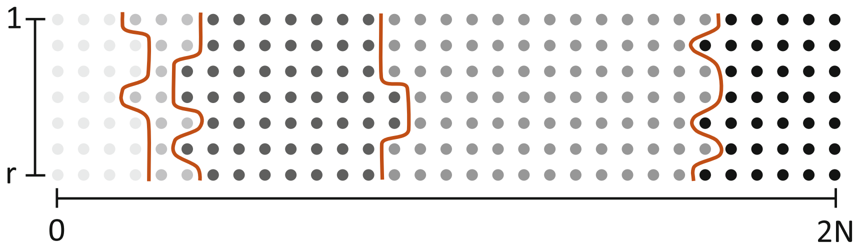

Fig. 2. Example probabilities $p_{i,x}$ when C contains 4 items, $r = 7$, and $N = 15$. The dots represent the various probabilities $p_{i,x}$, where rows are indexed by $i \in [r]$ and columns are indexed by $x \in [0, 2N]$. The shade of the dot at position (i, x) indicates the value of $p_{i,x}$, with darker shade indicating higher $p_{i,x}$. The elements in C describe curves from the top of the grid to the bottom, which are indicated in red in the figure. Notice (1) that the curves in C oscillate around *odd* columns of dots, and (2) that they never intersect, and (3) that the values of the $p_{i,x}$ only vary minimally between the curves in C, and can only have large changes when crossing the curves.

Proof. We prove the noiseless version, extending to the noisy version is a simple extension of Theorem 2. $\mathsf{PTrace}'^{P}(N, r, q, \delta)$ works as follows:

- First, we determine some of the s for elements in C. Let $P' : [0, N] \to [0, 1]_{\mathbb{R}}$ where $P'(x) = P(1, 2x)$. Notice that $|P'(N) - P'(0)| = |p_{2N} - p_0| > \epsilon$. Moreover, for intervals $(x, y]$ that do not contain any of the s, $|P'(y) - P'(x)| < \delta \le 2\delta$. Therefore, P' is an instance of the $(N, q, 2\delta, \epsilon)$ problem for $\epsilon > 2\delta(2 + (n-1)q)$. Therefore, we run $\mathsf{PTrace}^{P'}(N, q, \delta')$ to obtain a list $\mathcal{T}$ of s values, with the property that $|P(1, 2x) - P(1, 2x - 2)| = |P'(s) - P'(s-1)| \ge 2\delta$ for each $s \in \mathcal{T}$.
- For each $s \in \mathcal{T}$, and for each $i \in [r]$, let $b_{s,i} = 1$ if $|P(i, 2s-2) - P(i, 2s-1)| > |P(i, 2s-1) - P(i, 2s)|$, and $b_{s,i} = 0$ otherwise. Let $(s, b_1, \ldots, b_r) \in C$ be the tuple corresponding to s. Then the set C_i contains $2s - b_i$, but does not contain $2s - 1 + b_i$, since there is no collision between the s values. Therefore, $|P(2s - 1 + b_i) - P(2s - 2 + b_i)| < \delta$, which means that $|P(2s - b_i) - P(2s - 1 - b_i)| > \delta$. Therefore $b_{s,i} = b_i$
- Output the tuples $(s, b_{s,1}, \ldots, b_{s,r})$.

By the analysis above, since PTrace never outputs a value outside of C, PTrace' will never output a tuple corresponding to an identity outside of C. Moreover, if $\epsilon > \delta(4 + 2(n-1)q)$, then PTrace' will output at least one tuple in C. Finally, PTrace' runs in time only slightly worse than PTrace, and is therefore still polynomial time.

4 Tracing with Flexible Identities

Let $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a secure private *linear* broadcast scheme for identity space $\mathcal{ID} = [2^n]$. We now show that such a private broadcast scheme is sufficient for flexible traitor tracing. The Setup, KeyGen, Enc, and Dec algorithms are as follows:

- Setup, KeyGen are inherited from the private broadcast scheme.
- To encrypt a message m, run $\mathsf{Enc}(\mathsf{mpk}, S = \mathcal{ID}, m)$. Call this algorithm Enc_{TT}.
- To decrypt a ciphertext c, run $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c)$. Call this algorithm Dec_{TT}

Theorem 4. *Let* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a secure private broadcast scheme for identity space* $[2^n]$ *and private class* $\mathcal{S} = \{[u]\}_{u \in [0,2^n]}$. *Then there is a polynomial time algorithm* Trace *such that* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}_{TT}, \mathsf{Dec}_{TT}, \mathsf{Trace})$ *as defined above is a flexible traitor tracing algorithm.*

Proof. Boneh et al. [7] prove this theorem for the case of logarithmic n and for the weaker notion of tracing where the pirate decoder is required to decrypt a random message, as opposed to distinguish between two specific messages of the adversary's choice. Their tracing algorithm gets black-box access to a pirate decoder and does the following: it runs the decoder on encryptions to all sets $[u]$ for $u = 0, \ldots, 2^n$ and determines the success probability of the decoder for each u. It outputs an index u such that there is a "large" gap between the probabilities for $[u - 1]$ and $[u]$ as the identity of the traced traitor. In the analysis, [7] shows that, provided the adversary does not control the identity u, the pirate succeeds with similar probabilities for $[u - 1]$ and $[u]$. To prove this, they run the adversary, answering its secret key queries by making secret key queries to the PLBE challenger. When the adversary outputs a pirate decoder D, they make a PLBE challenge on a random message m and sets $[u]$ and $[u-1]$. Then they run the pirate decoder on the resulting ciphertext, and test whether it decrypts successfully: if yes, then they guess that the ciphertext was encrypted to $[u]$, and guess $[u - 1]$ otherwise. The advantage of this PLBE adversary is exactly the difference in probabilities for decrypting $[u-1]$ and $[u]$. The security of the PLBE scheme shows that this difference must be negligible.

Now, a useful pirate decoder will succeed with high probability on $[2^n]$, and with negligible probability on $[0]$, so there must be some "gap" in probabilities. The above analysis shows that (1) the tracer will find a gap, and (2) that the gap must occur at an identity under the adversary's control.

There are two problems with generalizing to our setting:

- The running time of the tracing algorithm in [7] grows with 2^n as opposed to n, resulting in an exponential-time tracing algorithm when using flexibly-large identities. This is because their tracing algorithm checks the pirate decoder an all identities. We therefore need a tracing algorithm that tests the decoder on a polynomial number of identities. To accomplish this, show that tracing amounts to solving the jump-finding problem in Sect. 3, and we can therefore use our efficient algorithm for the jump-finding problem to trace.
- Since we only ask that the pirate decoder can distinguish two messages, we need to reason about the decoder's "advantage" (decryption probability

minus 1/2) instead of its decryption probability. In the analysis above, since probabilities are always positive, any "useful" decoder will contribute positively to the PLBE advantage, whereas a "useless" decoder will not detract. However, this crucially relies on the fact that probabilities are positive. In our setting, the advantage is signed and can be both positive and negative, and the contribution of decoders to the PLBE adversary's advantage can cancel out if they have different sign. Thus there is no guarantee that the obtained PLBE adversary has any advantage. To get around this issue, we essentially have our reduction estimate the signed advantage of the pirate decoder, and reject all decoders with negative advantage. The result is that the advantage of all non-rejected decoders is non-negative, and so all decoders contribute positively to the PLBE adversary's advantage.

We now give our proof. Let $\mathcal{A}$ be a potential adversary, let C be the set of colluding parties for which $\mathcal{A}$ obtained secret keys, and $q = |C|$. $\mathcal{A}$ produces a pirate decoder $\mathcal{D}$ and messages m_0, m_1 such that $\mathcal{D}$ can distinguish encryptions of m_0 from encryptions of m_1. Define the quantities

$$p_{\mathsf{id}} = \Pr[\mathcal{D}(c) = b : b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m_b)]$$

for $\mathsf{id} \in \mathcal{S}$, where Enc is the PLBE encryption algorithm. We first will prove two lemmas:

Lemma 1. *Suppose* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure. Fix a non-negligible value* δ*. Suppose an interval* $(\mathsf{id}_L, \mathsf{id}_R]$ *is chosen adversarially after seeing the set* C*, the adversary's secret keys, the pirate decoder* $\mathcal{D}$*, and even the internal state of* $\mathcal{A}$*, and suppose that* $C \cap (\mathsf{id}_L, \mathsf{id}_R] = \emptyset$ *(that is, there are no colluding users in* $(\mathsf{id}_L, \mathsf{id}_R]$*). Then, except with negligible probability,* $|p_{\mathsf{id}_R} - p_{\mathsf{id}_L}| < \delta$*.*

Proof. We will prove that $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} < \delta$ with overwhelming probability, as proving $p_{\mathsf{id}_L} - p_{\mathsf{id}_R} < \delta$ is almost identical. Suppose towards contradiction that, with non-negligible probability ϵ, $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} \geq \delta$. We then describe an adversary for $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ that works as follows:

- Run $\mathcal{A}$ on input mpk. Whenever $\mathcal{A}$ makes a keygen query on identity id, make the same keygen query. $\mathcal{A}$ outputs a pirate decoder $\mathcal{D}$.
- Compute estimates $\hat{p_{\mathsf{id}_R}}, \hat{p_{\mathsf{id}_L}}$ for the probabilities p_{id_L} and p_{id_R}, respectively. To compute $\hat{p_{\mathsf{id}}}$, do the following. Take $O(\lambda/\delta^2)$ samples of $\mathcal{D}(c) \oplus b$ where $b \leftarrow \{0,1\}$ and $c \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m_b)$, and then output the fraction of those samples that result in 0. Notice that with probability $1-2^{-\lambda}$, $|\hat{p_{\mathsf{id}}} - p_{\mathsf{id}}| \leq \delta/4$.
- If $\hat{p_{\mathsf{id}_R}} - \hat{p_{\mathsf{id}_L}} < \frac{1}{2}\delta$, output a random bit and abort. Notice that, with overwhelming probability, $\left|(\hat{p_{\mathsf{id}_R}} - \hat{p_{\mathsf{id}_L}}) - (p_{\mathsf{id}_R} - p_{\mathsf{id}_L})\right| < \delta/2$. Therefore, with overwhelming probability, if we do not abort, $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} > 0$. Moreover, if $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} > \delta$, then $\hat{p_{\mathsf{id}_R}} - \hat{p_{\mathsf{id}_L}} \geq \frac{1}{2}\delta$ holds and we do not abort with overwhelming probability.
- Now choose a random bit b, and make a challenge query on $S_0^* = [\mathsf{id}_L]$, $S_1^* = [\mathsf{id}_R]$, and messages $m_0^* = m_1^* = m_b$.

– Upon receiving the challenge ciphertext c^*, compute $b' = \mathcal{D}(c^*)$. Output 1 if $b' = b$ and 0 otherwise.

Conditioned on no aborts, in the case the challenge ciphertext is encrypted to id_L (resp. id_R), our adversary will output 1 with probability p_{id_L} (resp. p_{id_R}), so our adversary will "win" with probability $\frac{1}{2} + (p_{\mathsf{id}_R} - p_{\mathsf{id}_L})/2$ in this case. Otherwise, during an abort, our adversary wins with probability 1/2. Moreover, with overwhelming probability, if we do not abort $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} > 0$, and with probability at least $\epsilon - \mathsf{negl}$, we have $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} > \delta/2$. Therefore, a simple computation shows that the adversary "wins" with probability at least $\frac{1}{2} + (\epsilon - \mathsf{negl})(\delta/4 - \mathsf{negl})$, which gives a non-negligible advantage.

Lemma 2. *Suppose* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure. Fix a non-negligible value* δ. *Then, except with negligible probability,* $|p_0 - \frac{1}{2}| < \delta$.

Proof. The proof is similar to the proof of Lemma 1. We will prove that $p_0 - \frac{1}{2} < \delta$ with overwhelming probability, the case $p_0 - \frac{1}{2} > -\delta$ is almost identical. Suppose towards contradiction that, with non-negligible probability ϵ, $p_0 - \frac{1}{2} \geq \delta$. An adversary for $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ works as follows:

– Run $\mathcal{A}$ on input mpk. Whenever $\mathcal{A}$ makes a keygen query on identity id, make the same keygen query. $\mathcal{A}$ outputs a pirate decoder $\mathcal{D}$.
– Compute estimate $\hat{p_0}$ for p_0 using the algorithm from Lemma 1, so that except with probability $2^{-\lambda}$, $|\hat{p_0} - p_0| < \delta/2$.
– If $\hat{p_0} - \frac{1}{2} < \frac{1}{2}\delta$, output a random bit and abort. Notice that, with overwhelming probability, $\left|(\hat{p_0} - \frac{1}{2}) - (p_0 - \frac{1}{2})\right| < \delta/2$. Therefore, with overwhelming probability, if we do not abort, $p_0 - \frac{1}{2} > 0$. Moreover, if $p_0 - \frac{1}{2} > \delta$, with overwhelming probability we do not abort.
– Now make a challenge query on $S_0^* = S_1^* = [0] = \{\}$, and messages $m_0^* = m_0, m_1^* = m_1$.
– Upon receiving the challenge ciphertext c^*, compute $b = \mathcal{D}(c^*)$. Output b

Conditioned on no aborts, our adversary will "win" with probability p_0 in this case. Otherwise, during an abort, our adversary wins with probability 1/2. Moreover, with overwhelming probability, if we abort $p_0 - \frac{1}{2} > 0$, and with probability at least $\epsilon - \mathsf{negl}$, we have $p_0 - \frac{1}{2} > \delta/2$. Therefore, a simple computation shows that the adversary has non-negligible advantage $(\epsilon - \mathsf{negl})(\delta/2 - \mathsf{negl})$.

Now we define our tracing algorithm $\mathsf{Trace}^{\mathcal{D}}(\mathsf{mpk}, m_0, m_1, q, \epsilon)$. Trace sets $\delta = \epsilon/2(5 + 4(n-2)q)$, and then runs $\mathsf{QTrace}^{Q}(2^n, q, \delta, \lambda)$ where QTrace is the algorithm from Theorem 2. Whenever QTrace makes a query to Q on identity id, Trace chooses a random bit b, computes the encryption $c \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m_b)$ of m_b to the set $[\mathsf{id}]$, runs $b' \leftarrow \mathcal{D}(c)$, and responds with 1 if any only if $b = b'$. Define p_{id} to be the probability that $Q(\mathsf{id})$ outputs 1. We now would like to show that Q is an instance of the (N, q, δ, ϵ) noisy jump finding problem, where the set of jumps is the set C. For this it suffices to show that $P(\mathsf{id}) = p_{\mathsf{id}}$ is an instance of the (N, q, δ, ϵ) *noiseless* jump finding problem. By Lemma 2, we have that with overwhelming probability useful $\mathcal{D}$ have $|p_{2^n} - p_0| \geq |\epsilon - \delta| > \epsilon/2$. Moreover, we have that $(\epsilon/2) = \delta(5 + 4(n-2)q)$.

Now we would hope that for any $(\mathsf{id}_L, \mathsf{id}_R]$ that do not contain one of the adversary's points, $|p_{\mathsf{id}_R} - p_{\mathsf{id}_L}| < \delta$. This would seem to follow from Lemma 1. However, we only have this property for $\mathsf{id}_L, \mathsf{id}_R$ that can be efficiently computed. Therefore, $P(\mathsf{id})$ is potentially a cheating oracle. However, since our tracing algorithm is efficient, any query it makes can be efficiently computed, and therefore $|p_{\mathsf{id}_R} - p_{\mathsf{id}_L}| < \delta$ holds (with overwhelming probability) for all *queried* points such that $(\mathsf{id}_L, \mathsf{id}_R]$ does not contain any of the identities in C. Therefore, following Remark 1, we can still invoke Theorem 2, which shows that the following hold:

- QTrace, and hence Trace, runs in polynomial time.
- QTrace, and hence Trace, will with overwhelming probability *not* output an identity outside S.
- If $\mathcal{D}$ is ϵ-useful, then QTrace, and hence Trace, will output *some* element in S (w.h.p.).

Construction. As observed by Garg et al. [21], FE immediately gives a PLBE scheme. Let $\mathcal{F}$ be the set of functions $f_{\mathsf{id}} : \mathcal{S} \times \mathcal{M} \to (\mathcal{M} \cup \{\perp\})$ where $f_{\mathsf{id}}(S, m)$ outputs m if $m \in S$ and $\perp$ if $m \notin S$. Let $(\mathsf{Setup}_{FE}, \mathsf{KeyGen}_{FE}, \mathsf{Enc}_{FE}, \mathsf{Dec}_{FE})$ be a FE scheme for this class of functions. The plaintext space $\mathcal{S} \times \mathcal{M}$ has size $2^\lambda \times |\mathcal{M}|$, and the function space admits circuits of size $O(\lambda)$. We then immediately obtain a PLBE scheme: to encrypt a message to a set S, simply encrypt the pair (S, m). The secret key for identity id is the secret key for function f_{id}. We use an adaptively secure scheme [2,21,40].

Parameter Sizes. In the above conversion, the PLBE scheme inherits the parameter sizes of the functional encryption scheme. Using functional encryption for general circuits, the secret size is $\mathsf{poly}(n)$ and the ciphertext size will similarly grow as $\mathsf{poly}(n, |m|)$. We can make the ciphertext size $|m| + \mathsf{poly}(n)$ by turning the PLBE into a key encapsulation protocol where we use the PLBE to encrypt the key for a symmetric cipher, and then encrypt m using the symmetric cipher. We note that it is inherent that the secret keys and ciphertexts of a PLBE scheme grow with the identity bit length n, as both terms must encode a complete identity. Therefore we obtain a PLBE scheme with essentially optimal parameters:

Corollary 1. *Assuming the existence of iO and OWF, then there exists an adaptively secure traitor tracing scheme whose master key is size is $O(1)$, secret key size is $\mathsf{poly}(n)$, and ciphertext size is $|m| + \mathsf{poly}(n)$.*

Note, however, that the obtained traitor tracing scheme is *not* optimal, as there is no reason ciphertexts in a traitor tracing scheme need to grow with the identity bit-length. The large ciphertexts are inherent to the PLBE approach to traitor tracing, so obtaining smaller ciphertexts necessarily requires a different strategy. In Sect. 5, we give an alternate route to obtaining traitor tracing that does not suffer this limitation, and we are therefore able to obtain an optimal traitor tracing system.

On Bounded Collusions. If we relax the security to bounded-collusion security, then the assumption can be relaxed to PKE using the q-bounded collusion FE scheme of [25].

Corollary 2. *Assume the existence of secure PKE, then there exists a q-bounded collusion-resistant adaptively secure traitor tracing scheme whose master key and secret key sizes are $O(n)\mathsf{poly}(q)$ and ciphertext size is $|m| + O(n)\mathsf{poly}(q)$.*

5 Flexible Traitor Tracing with Short Ciphertexts

We now discuss how to achieve traitor tracing with small ciphertexts that do not grow with the identity size. As noted above, the approach using private *linear* broadcast is insufficient due to having ciphertexts that inherently grow with the identity bit-length. We note that for traitor tracing, secret keys must encode the identities anyway, so they will always be as long as the identities. Therefore the focus here is just on obtaining short ciphertexts. To that end, we introduce a generalization of private linear broadcast that does not suffer from the limitations of the private linear broadcast approach; in particular, the information contained in the ciphertext is much shorter than the identities.

Let $\mathcal{ID}_0 = [2^{t+1}]$ be the set of identity "blocks", and the total identity space $\mathcal{ID} = (\mathcal{ID}_0)^n$ be the set of n-block tuples. Let $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a secure private broadcast scheme for $\mathcal{ID}$, and the secret class $\mathcal{S}$ defined as follows: each set $S_{i,u} \in \mathcal{S}$ is labeled by an index $i \in [n]$ and "identity block" $u \in \mathcal{ID}_0 \cup \{0\}$. $S_{i,u}$ is the set of tuples $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_n)$ where $\mathsf{id}_i \leq u$. We call such a private broadcast scheme a *private block linear broadcast encryption* (PBLBE) scheme.

Ideally, we would like to simply add a tracing algorithm on top of $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as we did in the previous section. The tracing algorithm would run the tracing algorithm from Sect. 4 on each identity block. For each $i \in [n]$, this gives a list of, say, T_i identity blocks $\mathsf{id}_{j,i} \in \mathcal{ID}_0$ for $j \in [T_i]$, where each of the $\mathsf{id}_{j,i}$ is the ith block of *some* identity owned by the adversary. Repeating this for every i gives a collection of identity blocks for every block number. However, it is not clear how to use these blocks to construct a complete identity in $\mathcal{ID}$. There are two problems:

- How do we argue that the blocks obtained for each index i come from the same set of identities? It may be that, for example when $n = 2$, that the adversary has identities $(\mathsf{id}_{1,1}, \mathsf{id}_{1,2})$ and $(\mathsf{id}_{2,1}, \mathsf{id}_{2,2})$, but tracing for $i = 1$ yields $\mathsf{id}_{1,1}$ whereas tracing $i = 2$ yields $\mathsf{id}_{2,2}$. While we have obtained two of the adversary's blocks, there may not even be a complete identity among the blocks.
- Even if we resolve the issue above, and show that tracing each block number yields blocks from the same set of identities, there is another issue. How to we match up the partial identity blocks? For example, in the case $n = 2$, we may obtain blocks $\mathsf{id}_{1,1}, \mathsf{id}_{2,1}, \mathsf{id}_{1,2}, \mathsf{id}_{2,2}$. However, we have no way of telling if the adversary's identities were $(\mathsf{id}_{1,1}, \mathsf{id}_{1,2})$ and $(\mathsf{id}_{2,1}, \mathsf{id}_{2,2})$, or if they were

$(\mathsf{id}_{1,1}, \mathsf{id}_{2,2})$ and $(\mathsf{id}_{2,1}, \mathsf{id}_{1,2})$. Therefore, while we can obtain the adversary's blocks for the adversary's identities, we cannot actually reconstruct the adversary's identities themselves.

We will now explain a slightly modified scheme and tracing algorithm to rectify the issues above. First, by including a fixed tag τ inside every block of id, we can now identify which blocks belong together simply by matching tags. This resolves the second point above, but still leaves the first. For this, we give a modified tracing algorithm that we can prove always outputs a complete collection of blocks.

We now give the scheme derived from any PBLBE. There will be two identity spaces. Let $\mathcal{ID}' = \{0,1\}^n$ be the identity space for the actual traitor tracing scheme; that is, $\mathcal{ID}'$ is the set of identities that we actually want to recover by tracing. We wish to grow n arbitrarily large without affecting the ciphertext size. The second space will be the space $\mathcal{ID}$ of the underlying PBLBE, which consists of n blocks of $t+1$ bits. In particular, the bit length of the traitor tracing identity space $\mathcal{ID}'$ will be equal to the number of blocks in the PBLBE space. Set $t = \lambda$, so that the bit-length of each block in the PBLBE grows with the security parameter, but crucially not in n. Define $N = 2^t = 2^\lambda$.

- Setup is again inherited from the private broadcast scheme.
- To generate the secret key for an identity $\mathsf{id}' \in \mathcal{ID}'$, write $\mathsf{id}' = (\mathsf{id}'_1, \ldots, \mathsf{id}'_n)$ where $\mathsf{id}'_i \in \{0,1\}$. Choose a random $s \in [N]$, and define the identity $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_n) \in \mathcal{ID}$ where $\mathsf{id}_i = 2s - \mathsf{id}'_i \in \mathcal{ID}_0$. Run the private broadcast keygen algorithm on id, and output the resulting secret key. Call this algorithm KeyGen_{TT}
- $\mathsf{Enc}, \mathsf{Dec}$ are identical to the basic tracing scheme, except that Dec now uses the derived user secret key as defined above. Call these algorithms $\mathsf{Enc}_{TT}, \mathsf{Dec}_{TT}$.

Theorem 5. *Let* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a secure private broadcast scheme for identity space* $\mathcal{ID}$ *and private class* $\mathcal{S}$*, where* $\mathcal{ID}, \mathcal{S}$ *are defined as above. Then there is an efficient algorithm* Trace *such that* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}_{TT}, \mathsf{Dec}_{TT}, \mathsf{Trace})$ *as defined above is a flexible traitor tracing algorithm.*

We prove Theorem 5 using similar techniques as in the proof of Theorem 4, except that the jump finding problem in Sect. 3 does not quite capture the functionality we need. Instead, in Sect. 3.1, we define a *generalized* jump finding problem, and show how to solve it. We then use the solution for the generalized jump finding problem to trace our scheme above.

Proof. We will take an approach very similar to the proof of Theorem 4. We will use a pirate decoder $\mathcal{D}$ to create an oracle Q as in the generalized jump finding problem. Then we run the tracing algorithm QTrace' on this Q, which will output the identities of some the colluders.

Define $Q(i,u)$ to be the randomized procedure that does the following: sample a random bit b, computes the encryption $c \leftarrow \mathsf{Enc}(\mathsf{mpk}, (i,u), m_b)$ of m_b to the set $S_{i,u}$ indexed by $(i,u) \in [n] \times [0,2N]$, runs $b' \leftarrow \mathcal{D}(c)$, and outputs 1 if and only if $b = b'$. Define $p_{i,u}$ to be the probability that $Q(i,u)$ outputs 1. We now need to show that if $\mathcal{D}$ is useful, then Q satisfies the conditions of Theorem 3.

First, notice that $p_{i,0} = p_{i',0}$ for all $i, i' \in [n]$, since the set indexed by $(i,0)$ is just the empty set, independent of i. Define $p_0 = p_{i,0}$. Similarly, $p_{i,2N} = p_{2N}$, independent of i, as the set indexed by $(i, 2N)$ is the complete set.

Next, notice that if $\mathcal{D}$ is useful, we have $|p_{2N} - p_0| > \epsilon/2$, similar to Theorem 4. Now set $\delta = \epsilon/(9+4(t-1)q)$ (recall that $N = 2^t$). We have the following:

Lemma 3. *Suppose* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure. Fix a non-negligible value δ. Suppose two pairs $(i,2x), (j,2x) \in [n] \times [0,2N]$ are chosen adversarially after seeing the set C, the adversary's secret keys, the pirate decoder $\mathcal{D}$, and even the internal state of $\mathcal{A}$. Then, except with negligible probability $|p_{i,2x} - p_{j,2x}| < \delta$*

Proof. Let id' be an identity the adversary queries on, with associated tag s. Let $\mathsf{id} = (\mathsf{id}_1, \dots, \mathsf{id}_n) \in \mathcal{ID}$ where $\mathsf{id}_i = 2s - \mathsf{id}'_i \in \mathcal{ID}_0$ as above. It suffices to show that the set $\mathsf{id} \in S_{i,2x}$ if and only if $\mathsf{id} \in S_{j,2x}$. This is equivalent to the requirement that $2s - \mathsf{id}'_i \leq 2x$ if and only if $2s - \mathsf{id}'_j \leq 2x$. Since $\mathsf{id}'_i, \mathsf{id}'_j$ are binary, this is true. The lemma then follows from the security of the private block linear broadcast scheme.

Next, define C_i to be the set of values $2s - \mathsf{id}'_i$ for identities id' queried by the adversary. Equivalently, C_i is the set of ith blocks of the corresponding identities id. The following also easily follows from the security of private block linear broadcast:

Lemma 4. *Suppose* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure. Fix a non-negligible value δ. Suppose two pairs $(i,x), (i,y) \in [n] \times [0,2N]$ are chosen adversarially after seeing the set C, the adversary's secret keys, the pirate decoder $\mathcal{D}$, and even the internal state of $\mathcal{A}$, such that the interval $(x,y]$ does not contain any points in C_i. Then $|p_{i,x} - p_{i,y}| < \delta$.*

We now see that the oracle Q corresponds to the $(N, r = n, q, \delta, \epsilon)$-generalized jump finding problem. Here, the hidden set C contains tuples $(s, \mathsf{id}_1, \dots, \mathsf{id}_n) = (s, \mathsf{id})$ where where $\mathsf{id} \in \mathcal{ID}'$ is one of the adversary's identities, and s is the corresponding tag that was used to generate the secret key for id. Similar to the basic tracing algorithm, the pirate decoder may *cheat*, and the lemmas above may not hold for all possible points. However, they hold for efficiently computable points, and in particular must hold for the points queried by the efficient QTrace' of Theorem 3. Thus, following Remark 1, we can invoke Theorem 3, so QTrace' will produce a non-empty list $\mathcal{L}$ of tuples (s, id) from C. This completes the theorem.

Construction and Parameter sizes. Similar to the case of PLBE, it is straightforward to construct private block linear broadcast encryption from functional encryption, and the PBLBE scheme will inherit the parameter sizes from the FE scheme. We will use $r = \lambda$-bit blocks and n-bit identities. The circuit size needed for the functional encryption scheme is therefore $\mathsf{poly}(n)$, and the plaintext size is $|m| + \mathsf{poly}(\log n)$ (ignoring the security parameter).

Some functional encryption schemes are non-compact, meaning the ciphertext size grows with both the plaintext size and the function size, in which case our ciphertexts will be $|m| + \mathsf{poly}(n)$, no better than the basic tracing system. Instead, we require compact functional encryption, where the ciphertext size is independent of the function size. The original functional encryption scheme of Garg et al. [21] has this property. However, they only obtain static security, and adaptive security is only obtained through complexity leveraging. In a very recent work, Ananth and Sahai [4] show how to obtain adaptively secure functional encryption for Turing machines, and in particular obtain adaptively secure functional encryption that meets our requirements for optimal ciphertext and secret key sizes.

Corollary 3. *Assuming the existence of iO and OWF, there exists an adaptively secure traitor tracing scheme whose master key size is* $\mathsf{poly}(\log n)$*, secret key size is* $\mathsf{poly}(n)$*, and ciphertext size is* $|m| + \mathsf{poly}(\log n)$*.*

On Bounded Collusions. If we relax security to bounded-collusion security, then the underlying assumption can be relaxed to the (sub-exponential) LWE assumption using the succinct FE scheme of [23], which can be made adaptively secure through complexity leveraging.

Corollary 4. *Assume the sub-exponential hardness of the LWE problem with a sub-exponential factor, then there exists a q-bounded collusion-resistant adaptively secure traitor tracing scheme whose master key size is* $\mathsf{poly}(\log n, q)$ *and secret key size is* $\mathsf{poly}(n, q)$ *and ciphertext size is* $|m| + \mathsf{poly}(\log n, q)$*.*

References

1. Abdalla, M., Dent, A.W., Malone-Lee, J., Neven, G., Phan, D.H., Smart, N.P.: Identity-based traitor tracing. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 361–376. Springer, Heidelberg (2007)
2. Ananth, P., Brakerski, Z., Segev, G., Vaikuntanathan, V.: From selective to adaptive security in functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 657–677. Springer, Heidelberg (2015)
3. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (2015)
4. Ananth, P., Sahai, A.: Functional encryption for turing machines. In: Kushilevitz, E., et al. (eds.) TCC 2016-A. LNCS, vol. 9562, pp. 125–153. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49096-9_6
5. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: Guruswami, V. (ed.) 56th Annual Symposium on Foundations of Computer Science, pp. 171–190. IEEE Computer Society Press, Berkeley, CA, USA, 17–20 October 2015

6. Boneh, D., Franklin, M.K.: An efficient public key traitor scheme (extended abstract). In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 338. Springer, Heidelberg (1999)
7. Boneh, D., Sahai, A., Waters, B.: Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 573–592. Springer, Heidelberg (2006)
8. Boneh, D., Waters, B.: A fully collusion resistant broadcast, trace, and revoke system. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 2006: 13th Conference on Computer and Communications Security, pp. 211–220. ACM Press, Alexandria, Virginia, USA, 30 October - 3 November 2006
9. Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 480–499. Springer, Heidelberg (2014)
10. Boyle, E., Chung, K.-M., Pass, R.: On extractability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 52–73. Springer, Heidelberg (2014)
11. Chabanne, H., Phan, D.H., Pointcheval, D.: Public traceability in traitor tracing schemes. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 542–558. Springer, Heidelberg (2005)
12. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 257–270. Springer, Heidelberg (1994)
13. Chor, B., Fiat, A., Naor, M., Pinkas, B.: Tracing traitors. IEEE Trans. Inf. Theor. **46**(3), 893–910 (2000)
14. Cohen, A., Holmgren, J., Nishimaki, R., Vaikuntanathan, V., Wichs, D.: Watermarking cryptographic capabilities. Cryptology ePrint Archive, Report 2015/1096 (2015). http://eprint.iacr.org/2015/1096
15. Cohen, A., Holmgren, J., Vaikuntanathan, V.: Publicly verifiable software watermarking. Cryptology ePrint Archive, Report 2015/373 (2015). http://eprint.iacr.org/2015/373
16. Dodis, Y., Fazio, N.: Public key broadcast encryption for stateless receivers. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 61–80. Springer, Heidelberg (2003)
17. Dodis, Y., Fazio, N.: Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 100–115. Springer, Heidelberg (2003)
18. Dodis, Y., Fazio, N., Kiayias, A., Yung, M.: Scalable public-key tracing and revoking. Distrib. Comput. **17**(4), 323–347 (2005)
19. Fiat, A., Tassa, T.: Dynamic traitor tracing. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 354. Springer, Heidelberg (1999)
20. Gafni, E., Staddon, J., Yin, Y.L.: Efficient methods for integrating traceability and broadcast encryption. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 372–387. Springer, Heidelberg (1999)
21. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th Annual Symposium on Foundations of Computer Science, pp. 40–49. IEEE Computer Society Press, Berkeley, CA, USA, 26–29 October 2013
22. Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure functional encryption without obfuscation. In: Kushilevitz, E., et al. (eds.) TCC 2016-A. LNCS, vol. 9563, pp. 480–511. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_18

23. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing, pp. 555–564. ACM Press, Palo Alto, CA, USA, 1–4 June 2013
24. Goodrich, M.T., Sun, J.Z., Tamassia, R.: Efficient tree-based revocation in groups of low-state devices. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 511–527. Springer, Heidelberg (2004)
25. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption with bounded collusions via multi-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 162–179. Springer, Heidelberg (2012)
26. Halevy, D., Shamir, A.: The LSD broadcast encryption scheme. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 47–60. Springer, Heidelberg (2002)
27. Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Roughgarden, T. (ed.) ITCS 2015: 6th Innovations in Theoretical Computer Science, pp. 163–172. Association for Computing Machinery, Rehovot, Israel, 11–13 January 2015
28. Kiayias, A., Tang, Q.: Traitor deterring schemes: using bitcoin as collateral for digital content. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015: 22nd Conference on Computer and Communications Security, pp. 231–242. ACM Press, Denver, CO, USA, 12–16 October 2015
29. Kiayias, A., Yung, M.: Traitor tracing with constant transmission rate. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 450–465. Springer, Heidelberg (2002)
30. Kiayias, A., Yung, M.: Copyrighting public-key functions and applications to black-box traitor tracing. Cryptology ePrint Archive, Report 2006/458 (2006). http://eprint.iacr.org/2006/458
31. Kurosawa, K., Desmedt, Y.G.: Optimum traitor tracing and asymmetric schemes. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 145–157. Springer, Heidelberg (1998)
32. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 41–62. Springer, Heidelberg (2001)
33. Naor, M., Pinkas, B.: Threshold traitor tracing. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 502–517. Springer, Heidelberg (1998)
34. Naor, M., Pinkas, B.: Efficient trace and revoke schemes. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 1–20. Springer, Heidelberg (2001)
35. Nishimaki, R., Wichs, D.: Watermarking cryptographic programs against arbitrary removal strategies. Cryptology ePrint Archive, Report 2015/344 (2015). http://eprint.iacr.org/2015/344
36. Nishimaki, R., Wichs, D., Zhandry, M.: Anonymous traitor tracing: how to embed arbitrary information in a key. Cryptology ePrint Archive, Report 2015/750 (2015). http://eprint.iacr.org/2015/750
37. Safavi-Naini, R., Wang, Y.: Sequential traitor tracing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 316–332. Springer, Heidelberg (2000)
38. Silverberg, A., Staddon, J., Walker, J.L.: Efficient traitor tracing algorithms using list decoding. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 175–192. Springer, Heidelberg (2001)
39. Tzeng, W.G., Tzeng, Z.J.: A public-key traitor tracing scheme with revocation using dynamic. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 207–224. Springer, Heidelberg (2001)

40. Waters, B.: A punctured programming approach to adaptively secure functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 678–697. Springer, Heidelberg (2015)
41. Zhandry, M.: Adaptively secure broadcast encryption with small system parameters. Cryptology ePrint Archive, Report 2014/757 (2014). http://eprint.iacr.org/2014/757