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

Ryo Nishimaki* Daniel Wichs† Mark Zhandry‡

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.

*NTT Secure Platform Laboratories, nishimaki.ryo@lab.ntt.co.jp. This work was done while the author was visiting Northeastern University.

†Northeastern University, wichs@ccs.neu.edu. 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.

‡Stanford University and MIT, mzhandry@gmail.com

1 Introduction

The Traitor-Tracing Problem. Traitor-tracing systems, introduced by Chor, Fiat and Naor [CFN94], 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 $[N] = \{1, \ldots, N\}$ 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 Tack 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 *flexibly large* (exponential size) universe of identities $\mathcal{ID}$ where a user's identity $\mathsf{id} \in \mathcal{ID}$ can 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 a traitor tracing scheme with a flexibly 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 its identity $\mathsf{id} \in \mathcal{ID}$ by sending id to the content provider directly, the user and the content provider can 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, Fiat and Naor [CFN94]. 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 [NP98, KD98, FT99, BF99, CFNP00, SW00, NP01, NNL01, SSW01, KY02, DF03, CPP05, BSW06, BW06, BZ14] 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 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). The work of Boneh, Sahai, and Waters [BSW06] builds 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. [GGH+13] and Boneh and Zhandry [BZ14] 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 [BSW06] and the main contributions are therefore the construction of a new PLBE scheme with the above parameters. These constructions both rely on indistinguishability obfuscation. More recently, Garg et al. [GGHZ14] construct a PLBE with polylogarithmic parameters based on simple assumptions on multilinear maps.

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., [DF02, GST04, HS02, NNL01, DF03, DFKY05, GSY99, NP01, TT01] and references within.

A trace and revoke system is a combination of broadcast encryption and traitor tracing [NP01, NNL01]. In other words, once traitors are identified by the tracing algorithm they can also be revoked from decrypting future ciphertexts. Boneh and Waters [BW06] 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 poly-logarithmic parameter sizes. Separately, though, it is known how to build both broadcast encryption and traitor tracing with such parameters using obfuscation [Zha14, GGH+13, BZ14], 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 [NW15, CHV15]. 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 $\mathsf{poly}(\log n)$. The scheme is secure for

unbounded collusions assuming the existence of indistinguishability obfuscation (iO) and one-way functions (OWF). 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 public-key encryption, or of length $\mathsf{poly}(\log n, q)$ assuming sub-exponential LWE.[1] 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 as 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, Sahai, and Waters [BSW06], 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 and more interesting challenge 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 [BZ14] already constructs a PLBE scheme where the key/ciphertext size is linear 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. We instead take a different approach, suggested by [GGH+13], and construct PLBE directly from (indistinguishability based) functional encryption (FE). For technical reasons detailed below, we 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 [Wat14, ABSV14] or from simple assumptions on multilinear maps [GGHZ14]. 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 [GVW12] or from sub-exponential LWE if we want succinct ciphertexts [GKP+13].

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 $T \in \{0, \ldots, N\}$ 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 $i \in \{T, \ldots T' - 1\}$, 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 [BSW06] 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. Our goal is to design a better algorithm that takes at most $\mathsf{poly}(n, q)$ steps.

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

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, Chung and Pass [BCP14] 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 [BSW06], for two reasons: (1) their tracing algorithm visits *all* $T \in [N]$, and (2) in the case of polynomial N statically secure and adaptive secure PLBE are equivalent. Fortunately for us, 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 [GVW12].

Tracing More General Decoders. In [BSW06], 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 [BSW06] appears insufficient for this setting. In our analysis, we show that even if a decoder can distinguish between two particular messages 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. Roughly, we then trace the value contained in each block one at a time. This lets use 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 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 [AS15] shows how to build such an FE from iO. In the bounded collusion setting, we can obtain such an FE from LWE using [GKP+13], 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 show how to construct such a scheme from indistinguishability obfuscation using the technique of somewhere statistically binding (SSB) hashing [HW15].

1.4 Outline

In Section 2, we give some definitions and notations that we will use in our work. In Section 3, we define the oracle jump-finding problem, and show how to efficiently solve it. In Sections 4 and 5, we use the solution of the jump-finding problem to give our new traitor tracing schemes. In Section 6, we give our full trace and revoke scheme.

2 Preliminaries

2.1 Notations

Throughout this work, we will use the notation $[N]$ to mean the positive integers from 1 to N: $[N] = \{1, \ldots, N\}$. 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.

2.2 Traitor Tracing with Flexible Identities

Here we define traitor tracing. Our definition is similar to that of Boneh, Sahai, and Waters [BSW06], 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 [BSW06] only allows for tracing pirate decoders that can decrypt encryptions of random messages. In Section 4, we discuss why the analysis in [BSW06] appears insufficient for our more general setting, but nevertheless show that tracing is still possible.

Definition 2.1. 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

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

- **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. We say 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 is 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 $\Pr[\mathcal{D}(c) = m_b^* : b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b^*)] \geq \frac{1}{2} + \epsilon$[2].
  · $\mathcal{T}$ 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.

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.

2.3 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, Sahai, and Waters [BSW06]. 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 Section 5

Definition 2.2. 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

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

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

  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 $\perp$.

[2]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 produce. 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.

- **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 is 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.
  - Finally, $\mathcal{A}$ 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 $|\mathsf{sk}_{\mathsf{id}}| \geq \mathsf{poly}(\lambda, \log |\mathcal{ID}|)$.

- **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.

2.4 Obfuscation

The notion of indistinguishability obfuscation (iO) was proposed by Barak et. al. [BGI+01, BGI+12] and the first candidate construction was proposed by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13].

Definition 2.3 (Indistinguishability Obfuscation [BGI+01, BGI+12, GGH+13])**.** A PPT algorithm $\mathcal{O}$ is an indistinguishability obfuscator (iO) if it satisfies the following two conditions.

Functionality: For all security parameter $\lambda \in \mathbb{N}$, for all circuit C for all input x, it holds that

$$\Pr[C'(x) = C(x) \mid C' \leftarrow \mathcal{O}(1^\lambda, C)] = 1.$$

Indistinguishability: For all PPT distinguisher $\mathcal{D}$ and all circuit ensembles $C_0 = \{C_\lambda^{(0)}\}_{\lambda\in\mathbb{N}}$ and $C_1 = \{C_\lambda^{(1)}\}_{\lambda\in\mathbb{N}}$ such that $\forall\lambda, x : C_\lambda^{(0)}(x) = C_\lambda^{(1)}(x)$ and $|C_\lambda^{(0)}| = |C_\lambda^{(1)}|$ there exists a negligible function α, we have:

$$\left|\Pr[\mathcal{D}(\mathcal{O}(1^\lambda, C_\lambda^{(0)})) = 1] - \Pr[\mathcal{D}(\mathcal{O}(1^\lambda, C_\lambda^{(1)})) = 1]\right| < \alpha(\lambda)$$

For simplicity, we write $\mathcal{O}(C)$ instead of $\mathcal{O}(1^\lambda, C)$ when the security parameter λ is clear from context.

Puncturable Pseudorandom Functions. The notion of puncturable pseudorandom function (pPRF) was proposed by Sahai and Waters [SW14]. This is a useful tool to prove the security of cryptographic schemes based on iO.

Definition 2.4 (Puncturable Pseudorandom Functions)**.** For sets D, R, a puncturable pseudorandom function (pPRF) $\mathcal{F}$ consists of a tuple of (probabilistic) algorithms $\mathcal{F} = (\mathsf{PRF.Gen}, \mathsf{F}, \mathsf{Punc})$ that satisfy the following two conditions.

Functionality preserving under puncturing: For all polynomial size set $S \subseteq \mathsf{D}$ and for all $x \in \mathsf{D} \setminus S$, it holds that

$$\Pr[\mathsf{F}_r(x) = \mathsf{F}_{r\{S\}}(x) \mid r \leftarrow \mathsf{PRF.Gen}(1^\lambda), r\{S\} = \mathsf{Punc}(r, S)] = 1.$$

Pseudorandom at punctured points: For all polynomial size set $S = \{x_1, \ldots, x_{k(\lambda)}\} \subseteq \mathsf{D}$ and all PPT distinguisher $\mathcal{D}$, there exists negligible function α, we have:

$$\left|\Pr[\mathcal{D}(\mathsf{F}_{r\{S\}}, \{\mathsf{F}_r(x_i)\}_{i\in[k]}) = 1] - \Pr[\mathcal{D}(\mathsf{F}_{r\{S\}}, U^k) = 1]\right| \leq \alpha(\lambda)$$

where $r \leftarrow \mathsf{PRF.Gen}(1^\lambda)$, $r\{S\} = \mathsf{Punc}(r, S)$ and U denotes the uniform distribution over R.

Theorem 2.5 ([GGM86, BW13, BGI14, KPTZ13])**.** *If one-way functions exits, then for all efficiently computable $n(\cdot)$ and $m(\cdot)$, there exists a pPRF family whose input is an $n(\cdot)$ bit string and output is an $m(\cdot)$ bit string.*

2.5 Functional Encryption

Definition 2.6. 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

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

- **Security.** Intuitively, we ask that the adversary, given secret keys $f_1, \ldots, f_n$, leans $f_i(m)$ 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 $f(m_0^*) = f(m_1^*)$.
  - Finally, $\mathcal{A}$ 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 $|c| \leq \mathsf{poly}(\lambda, \log |\mathcal{M}|)$[3].

[3]This property has been referred to as "compactness" [AJ15, BV15].

- **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 [AS15]. The construction is based on indistinguishability obfuscation for circuits and one-way function.

2.6 Symmetric Key Encryption with Pseudorandom Ciphertexts

Definition 2.7. Let $\mathcal{M}$ be some message space. A symmetric key encryption scheme for $\mathcal{M}$ is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ where:

- $\mathsf{KeyGen}()$ is a randomized procedure with no input (except the security parameter) that outputs a secret key k.
- $\mathsf{Enc}_k(m)$ takes as input the secret key k and a message $m \in \mathcal{M}$, and outputs a ciphertext c.
- $\mathsf{Dec}_k(c)$ takes as input the secret key k and a ciphertext c, and outputs some $m \in \mathcal{M}$, or $\perp$.
- **Correctness.** For any message $m \in \mathcal{M}$, we have that

$$\Pr[\mathsf{Dec}_k(c) = m : k \leftarrow \mathsf{KeyGen}(), c \leftarrow \mathsf{Enc}_k(m)] = 1$$

- **Security.** The security is formalized by the following experiment between an adversary $\mathcal{A}$ and challenger:
  - The challenger runs $k \leftarrow \mathsf{KeyGen}()$ and flips a random bit $b \in \{0, 1\}$.
  - $\mathcal{A}$ is allowed to make arbitrary ciphertext queries, where it sends a message $m \in \mathcal{M}$ to the challenger. If $b = 0$, then the challenger responds with $\mathsf{Enc}_k(m)$. If $b = 1$, then the challenger outputs a uniformly random string of length $\ell(\lambda)$ where $\ell(\lambda)$ is the length of the ciphertexts.
  - Finally, $\mathcal{A}$ 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 symmetric key encryption scheme with pseudorandom ciphertexts is secure if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.

We can construct such a symmetric encryption scheme with pseudorandom ciphertexts from OWF. For example, we can use $\mathsf{Enc}_k(m; r) = (r, \mathsf{F}_k(r) \oplus m)$ where F_k is a standard PRF with a key k and r is a uniformly random string (randomness for the ciphertext) [Gol04].

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 [BCP14] as well as the tracing algorithm in this work.

Definition 3.1. 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 Figure 1.

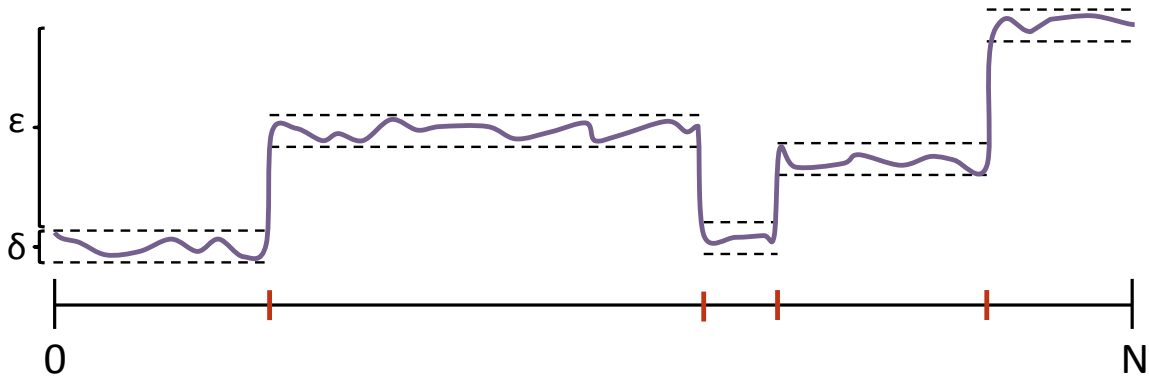

Figure 1: Example of an oracle P when C contains 4 points. The purple curve represents the outputs of the oracle P on inputs in the interval $[0, N]$. The red hatch marks on the number line indicate the positions of the elements in C. The horizontal dashed lines show that, between the points in C, P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

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 impossible to determine an $x \in C$ exactly. 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. Therefor, 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:

Theorem 3.2. *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 that* $\epsilon \geq \delta(2 + (\lceil \log N \rceil - 1)q)$*. 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.

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 $|I| = b - a$ and let q_I be the number of points in C in the interval 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 3.3. *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 3.4. *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 3.5. *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. □

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. □

Remark 3.6. We note that PTrace^P works 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. This will be crucial in our traitor tracing construction.

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

Definition 3.7. The (N, q, δ, ϵ) noisy jump finding problem is the following. An adversary chooses a set $C \subseteq [1, N]$ of q unknown points. Then, the adversary builds an oracle $P : [0, N] \to [0, 1]_{\mathbb{R}}$ as above, but does not provide it to us 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 3.8. *There is a probabilistic algorithm* $\mathsf{QTrace}^O(N, q, \delta, \lambda)$ *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 $\hat{P}$.

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

- For $i = 1, \ldots, O(\lambda/\delta^2)$, sample $z_i \leftarrow O(x)$
- Output $\hat{p}_x$ as the average 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 $(N, q, \delta' = 2\delta, \epsilon' = \epsilon - \delta)$. 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 3.2, and PTrace outputs at least one element in C. QTrace outputs the same element, completing the proof. □

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 points in C are more complex, and consist of a tag $s \in [N]$, and a bit-string $b = (b_1, \ldots, b_r) \in \{0,1\}^r$. Our goal is to determine both s and b for some pair $(s, b) \in C$. However, unlike the previous jump finding problem, we are not given an oracle that allows for tracing both terms directly. Instead, we are given r separate jump-finding oracles corresponding to sets $C_i = \{2s - b_i\}_{(s,b)\in C} \subseteq [2N]$. Moreover, we are given that the probabilities underlying the different oracles are very close.

Definition 3.9. The $(N, r, q, \delta, \epsilon)$ generalized jump finding problem is the following. There is a set $C \subseteq [1, N]$ of q unknown tuples $(s, b_1, \ldots, b_r) \in [N] \times \{0,1\}^r$ such that the s are distinct. Let C_i be the set of points $2s - b_i$ for tuples in C. For each pair $(i, x) \in [1, r] \times [0, 2N]$ there is a probability $p_{i,x} \in [0,1]_{\mathbb{R}}$ with the properties that:

- $p_{i,0} = p_{i',0}$ for all $i, i' \in [m]$. Similarly, $p_{i,N} = p_{i',N}$ for all $i, i' \in [r]$. Define $p_0 = p_{i,0}$ and $p_N = p_{i,N}$.
- $|p_N - p_0| > \epsilon$
- For any pair of pairs of the form $(i, 2x), (j, 2x) \in [1, r] \times [0, 2N]$, $|p_{i,2x} - p_{j,2x}| < \delta$
- For any pair of pairs of the form $(i, x), (i, y) \in [1, r] \times [0, 2N]$ such that the interval $(x, y]$ does not contain any points in C_i (that is, $(x, y] \cap C_i = \emptyset$), then $|p_{i,x} - p_{i,y}| < \delta$.

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

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

A pictorial representation of the generalized jump finding problem is given in Figure 2.

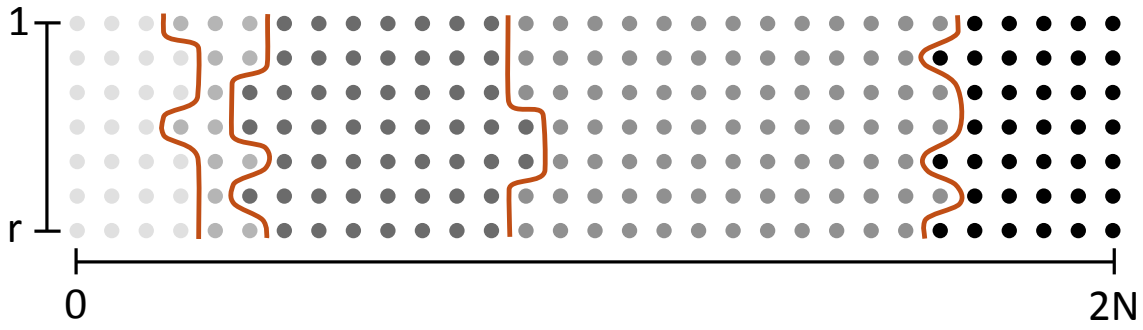

Figure 2: Example probabilities $p_{i,x}$ when C contains 4 items, $r = 7$, and $N = 15$. The dots represent the various probabilities $p_{i,x}$, where rows are indexed by $i \in [r]$ and columns are indexed by $x \in [0, 2N]$. The shade of the dot at position (i, x) indicates the value of $p_{i,x}$, 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.

Theorem 3.10. *There is 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* $(N, r, q, \delta, \epsilon)$ *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 3.2 and 3.8, and appears in below. Again, PTrace', 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'.

Proof. We prove the noiseless version, extending to the noisy version is a simple extension of Theorem 3.8. $\mathsf{PTrace}'^{P}(N, r, q, \delta)$ works as follows:

- Let $P'(x) = P(1, 2x)$. Notice that $|P'(N) - P'(0)| = |p_N - p_0| > \epsilon$. Moreover, for intervals $(x, y]$ that do not contain any of the s, $|P'(y) - P'(x)| < \delta \leq 2\delta$. 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 $|P(1, 2x) - P(1, 2x-2)| = |P'(s) - P'(s-1)| \geq 2\delta$ for each $s \in \mathcal{T}$.
- For each $s \in \mathcal{T}$, and for each $i \in [n]$, 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 4.1. *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, Sahai, and Waters [BSW06] 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. 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, [BSW06] 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 [BSW06] 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 Section 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

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

for $\mathsf{id} \in \mathcal{S}$. We first will prove two lemmas:

Lemma 4.2. *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 $p_{\hat{\mathsf{id}}_R}, p_{\hat{\mathsf{id}}_L}$ for the probabilities p_{id_L} and p_{id_R}, respectively. To compute $p_{\hat{\mathsf{id}}}$, 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 $1 - 2^{-\lambda}$, $|p_{\hat{\mathsf{id}}} - p_{\mathsf{id}}| \leq \delta/4$.
- If $p_{\hat{\mathsf{id}}_R} - p_{\hat{\mathsf{id}}_L} < \frac{1}{2}\delta$, output a random bit and abort. Notice that, with overwhelming probability, $|(p_{\hat{\mathsf{id}}_R} - p_{\hat{\mathsf{id}}_L}) - (p_{\mathsf{id}_R} - p_{\mathsf{id}_L})| < \delta/2$. Therefore, with overwhelming probability, if we do not abort, $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} > 0$. Moreover, if $p_{\mathsf{id}_R} - p_{\mathsf{id}_L} > \delta$, with overwhelming probability we do not abort.
- 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 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 4.3. *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 4.2. 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 4.2, 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, $|(\hat{p_0} - \frac{1}{2}) - (p_0 - \frac{1}{2})| < \delta/2$. 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 3.8. 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 4.3, 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 4.2. 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 3.6, we can still invoke Theorem 3.8, 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. [GGH+13], 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 [GGH+13, ABSV14, Wat14].

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 4.4. *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 Section 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 [GVW12].

Corollary 4.5. *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 in the previous section, the approach based on 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 $\mathcal{ID}_0 = [2^{r+1}]$ 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 $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_r)$ 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 Section 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. Let $\mathcal{ID}' = \{0,1\}^n$ be the traitor tracing identity space, and set $r = \lambda$.

- Setup is again inherited from the private broadcast scheme.
- To generate the secret key for an identity $\mathsf{id}' \in \mathcal{ID}'$, write $\mathsf{id}' = (\mathsf{id}'_1, \ldots, \mathsf{id}'_r)$ where $\mathsf{id}'_i \in \{0,1\}$. Choose a random $s \in [2^\lambda]$, and define the identity $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_r) \in \mathcal{ID}$ 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 5.1. *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 5.1 using similar techniques as in the proof of Theorem 4.1, except that the jump finding problem in Section 3 does not quite capture the functionality we need. Instead, in Section 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 4.1. 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 indexed by (i, u), 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 3.10.

First, notice that $p_{i,0} = p_{i',0}$ for all $i, i' \in [r]$, since the set indexed by $(i, 0)$ is just the empty set, independent of i. Define $p_0 = p_{i,0}$. Similarly, $p_{i,2^{n+1}} = p_{2^{n+1}}$, independent of i, as the set indexed by $(i, 2^{n+1})$ is the complete set.

Next, notice that if $\mathcal{D}$ is useful, we have $|p_{2^{n+1}} - p_0| > \epsilon/2$, as in Theorem 4.1. Now set $\delta = \epsilon/(9 + 4(n-1)q)$. We have the following:

Lemma 5.2. *Suppose* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure. Fix a non-negligible value* δ*. Suppose two pairs* $(i, 2x), (j, 2x) \in [1, r] \times [0, 2N]$ *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 $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_n) \in \mathcal{ID}$ 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 the private block linear broadcast scheme:

Lemma 5.3. *Suppose* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is secure. Fix a non-negligible value* δ*. Suppose two pairs* $(i, x), (i, y) \in [1, r] \times [0, 2N]$ *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* $|p_{i,x} - p_{i,y}| < \delta$*.*

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 QTrace'. Thus, following Remark 3.6, we can invoke Theorem 3.10, so QTrace' will produce a list $\mathcal{L}$ of tuples $(s, \mathsf{id}'_1, \ldots, \mathsf{id}'_r)$, where id' is an identity queries by the adversary with corresponding tag s. 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. [GGH+13] 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 [AS15] 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 5.4. *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 [GKP+13], which can be made adaptively secure through complexity leveraging.

Corollary 5.5. *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* $|m| + \mathsf{poly}(\log n, q)$.

6 A Flexible Trace and Revoke Scheme

In this section, we propose a flexible trace and revoke scheme with short ciphertexts. We can construct it from PBE *with revocation* and short ciphertexts as the flexible traitor tracing scheme from PBE. To achieve PBE with revocation and short ciphertexts, we introduce a notion of revocable functional encryption and propose a construction based on functional encryption and indistinguishability obfuscation.

6.1 Revocable Functional Encryption

We start with the definition of a revocable functional encryption scheme. As with functional encryption, this scheme can be used to give out secret keys for various functions f, but we now also bind each secret key to some (short) label lb, denoted by $\mathsf{sk}_{\mathsf{lb},f}$. During the encryption of a message m, the user can also specify a list L of revoked labels. The decryption procedure using $\mathsf{sk}_{\mathsf{lb},f}$ should recover $f(m)$ if $\mathsf{lb} \notin L$. On the other hand, if $\mathsf{lb} \in L$, then the key $\mathsf{sk}_{\mathsf{lb},f}$ should not provide any additional information. We sketch the definition below.

Definition 6.1. Let $\mathcal{M}$ be some message space, $\mathcal{Y}$ the output space, $\mathcal{F}$ be a class of functions $f : \mathcal{M} \to \mathcal{Y}$, and $\mathcal{L}$ be a label space. A Revocable Functional Encryption (revFE) scheme for $\mathcal{M}, \mathcal{Y}, \mathcal{F}, \mathcal{L}$ 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.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, f)$ takes as input the master secret msk, a label $\mathsf{lb} \in \mathcal{L}$ and a function $f \in \mathcal{F}$. It outputs a secret key $\mathsf{sk}_{\mathsf{lb},f}$ for f.
- $\mathsf{Enc}(\mathsf{mpk}, m, L)$ takes as input the master public key mpk, a message $m \in \mathcal{M}$ and a revocation list $L \subseteq \mathcal{L}$. Outputs a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{lb},f}, L, \mathsf{lb}, c)$ takes as input the secret key $\mathsf{sk}_{\mathsf{lb},f}$ and a ciphertext c, and outputs some $y \in \mathcal{Y}$, or $\perp$.

- **Correctness.** For any message $m \in \mathcal{M}$, any set $L \subseteq \mathcal{L}$, any $\mathsf{lb} \notin L$ and any function $f \in \mathcal{F}$ and we have that

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

- **Security.** Intuitively, we ask that the adversary, given secret keys $f_1, \ldots, f_n$ for labels $\mathsf{lb}_1, \ldots, \mathsf{lb}_n$ learns $f_i(m)$ for each i such that $\mathsf{lb}_i \notin L$, but nothing else about m. Although the challenge message and the secret key queries are all adaptive, our security definition requires that the revoke set S is specified before secret key queries. Security is formalized by the following experiment between an adversary $\mathcal{A}$ and challenger:

  1. The challenger runs $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$, and gives mpk to $\mathcal{A}$.
  2. The attacker $\mathcal{A}$ specifies a "revoke set" $L \subseteq \mathcal{L}$.
  3. $\mathcal{A}$ is allowed to make arbitrary key queries, where it sends a pair $(\mathsf{lb}, f) \in (\mathcal{L}, \mathcal{F})$ to the challenger, and the challenger responds with $\mathsf{sk}_{\mathsf{lb},f} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, f)$. The challenger also records the pair (lb, f) in a list $\mathcal{R}$ and checks that each query has a fresh lb.
  4. 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 pairs $(f, \mathsf{lb}) \in \mathcal{R}$ such that $\mathsf{lb} \notin L$. If the check fails 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 $c^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b^*, L)$.
  5. $\mathcal{A}$ is allowed to make additional keygen queries for functions $(\mathsf{lb}, f) \in \mathcal{F}$, subject to the constraint that either $\mathsf{lb} \in L$ *or* $f(m_0^*) = f(m_1^*)$. As before, the challenger also records the pair (lb, f) in a list $\mathcal{R}$ and checks that each query has a fresh lb.
  6. Finally, $\mathcal{A}$ 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.

- **Ciphertext size.** We will consider a scheme to have *optimal* ciphertext size if $|c| \leq \mathsf{poly}(\lambda, \log |\mathcal{M}|)$ and independent of L.[4].

- **Secret key size.** The secret key must information-theoretically encode the entire function f, so $|\mathsf{sk}_f| \leftarrow \log |\mathcal{F}|$. 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 $|\mathsf{msk}|, |\mathsf{mpk}| \leq \mathsf{poly}(\lambda)$.

Remark 6.2. If ciphertext size can grow with size of revoke list L, then there is a simple construction of revocable FE from standard FE. The revocable key-generation procedure for (lb, f) would run the key-generation procedure of the standard FE scheme with the function $f_{\mathsf{lb}}(m, L)$ defined as $f_{\mathsf{lb}}(m, L) = f(m)$ if $\mathsf{lb} \notin L$ and $\perp$ otherwise. The revocable encryption procedure would just encrypt the pair (m, L). Our goal will be to remove the dependence on the size of the revoke list.

[4]This property has been referred to as "compactness" [AJ15, BV15].

6.2 Tool for Revocable FE: SSB hash on Sets

Definition 6.3. A *Somehwere Statistically Binding (SSB) Hash on Sets* $\mathcal{H}$ consists of four PPT algorithms $\mathcal{H} = (\mathsf{Gen}, H, \mathsf{Open}, \mathsf{Verify})$ an output size $\ell(\cdot,\cdot)$ and an opening size $p(\cdot,\cdot)$. The algorithms have the following syntax:

- $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i)$: Takes as input a security parameter λ an element-size s and an index $i \in \{0,1\}^s$ and outputs a public hashing key hk. The output size is defined by some fixed polynomial $\ell = \ell(\lambda, s)$.
- $H_{\mathsf{hk}} : Powerset(\{0,1\}^s) \to \{0,1\}^\ell$: A deterministic poly-time algorithm that takes as input $S \subseteq \{0,1\}^s$ and outputs $H_{\mathsf{hk}}(S) \in \{0,1\}^\ell$.
- $\pi \leftarrow \mathsf{Open}(\mathsf{hk}, S, j)$: Given the hash key hk, $S \subseteq \{0,1\}^s$ and an index $j \in \{0,1\}^s$, creates an opening $\pi \in \{0,1\}^p$ that certifies whether or not $j \in S$. The opening size $p = p(\lambda, s)$ is some fixed polynomial in λ, s.
- $\mathsf{Verify}(\mathsf{hk}, y, j, b, \pi)$: Given a hash key hk a hash output $y \in \{0,1\}^\ell$, an index $j \in \{0,1\}^s$, a bit b and an opening $\pi \in \{0,1\}^p$, outputs a decision $\in \{\mathsf{accept}, \mathsf{reject}\}$. This is intended to verify that $y = H_{\mathsf{hk}}(S)$ correspond to a set S that either has $j \in S$ if $b = 1$ or not if $b = 0$.

We require the following three additional properties.

Correctness of Opening: For any parameters s and any indices $i, j \in \{0,1\}^s$, any $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i)$, $S \subseteq \{0,1\}^s$, $\pi \leftarrow \mathsf{Open}(\mathsf{hk}, S, j)$: we have $\mathsf{Verify}(\mathsf{hk}, H_{\mathsf{hk}}(S), j, b, \pi) = \mathsf{accept}$ where $b = 1$ if $j \in S$ and 0 otherwise.

Index Hiding: We consider the following experiment between an adversary $\mathcal{A}$ and challenger:

1. $\mathcal{A}$ chooses an element-size s and two indices $i_0, i_1 \in \{0,1\}^s$ and send them to the challenger.
2. The challenger chooses a bit $b \leftarrow \{0,1\}$ and generates $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i_b)$.
3. $\mathcal{A}$ is given hk and 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 SSB hash is index hiding if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.

Somewhere Statistically Binding w.r.t. Opening: We say that hk is *statistically binding w.r.t opening for an* index i if there do not exist y, π, π' such that $\mathsf{Verify}(\mathsf{hk}, y, i, 1, \pi) = \mathsf{Verify}(\mathsf{hk}, y, i, 0, \pi') = \mathsf{accept}$. We require that for any parameters s, L and any index $i \in \{0,1\}^s$

$$\Pr[\mathsf{hk} \text{ is statistically binding w.r.t. opening for index } i \ : \ \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i)] \geq 1 - \mathsf{negl}(\lambda).$$

We say that the hash is *perfectly binding w.r.t. opening* if the above probability is 1.

Construction. We can easily adapt the SSB hash of [HW15] to the above setting. The main difference is that, instead of hashing a polynomial-sized string, we now need to hash a polynomial-sized subset S of an exponential universe $S \subseteq \{0,1\}^s$. We can think of S as being represented by a sparse string w_S of length $|w_S| = 2^s$ that has a 1 for each element in S and a 0 otherwise. The SSB hash of [HW15] allows us to hash such exponential sized sparse strings. It follows a Merkle-Tree structure where the leaves correspond to the

bits of the hashed string and the value associated with each node is computed inductively as some function of the values associated with its children - the output of the hash is the value associated with the root. We can use this structure to efficiently hash an exponentially long sparse string w_S by simply thinking of an exponentially large tree of height s but "pruning out" all sub-trees that just have 0's under them – the root of any such sub-tree now becomes a leaf with the value 0. The pruned tree is now an unbalanced tree of height s and polynomial size. All procedures of [HW15] can be adapted to work with such unbalanced Merkle-Trees.

6.3 Constructing Revocable FE

We show how to construct revocable FE scheme $\mathcal{F}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$. We rely on an adaptively secure FE scheme $\mathcal{F} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ (See Definition 2.5), an SSB hash $\mathcal{H} = (\mathsf{Gen}, H, \mathsf{Open}, \mathsf{Verify})$, a secure symmetric-key encryption with pseudorandom ciphertexts scheme $\mathcal{E} = (\mathsf{SymGen}, \mathsf{SymEnc}, \mathsf{SymDec})$, a puncturable PRF $\mathcal{G}$ and iO scheme $\mathcal{O}$. We use an extension of Trojan method [ABSV14].

- $\mathsf{Setup}'()$ is the same as Setup.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, f)$: Define the functions lock and wrap as shown.

$\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, z](\pi)$: Output z if $\mathsf{Verify}(\mathsf{hk}, y, \mathsf{lb}, 0, \pi) = \mathsf{accept}$ and $\perp$ otherwise.

$\mathsf{wrap}[\mathsf{lb}, f, c_{sym}](\mathsf{hk}, y, r, m, \mathsf{flag}, \mathsf{aux} = (i, m', r', k', v))$
if $\mathsf{flag} = 0$ **then** ▷ Real branch
  Output $\mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m)]; \mathsf{F}_r(\mathsf{lb}))$;
else ▷ Trojan branches
  Compute $(j, b_{rev}, b_{pre}, w) \leftarrow \mathsf{SymDec}_{k'}(c_{sym})$;
  if $b_{pre} = 0$ **then**
    Output w; ▷ Trojan branch (1)
  else if $j > i$ **then**
    Output $\mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m)]; \mathsf{F}_r(\mathsf{lb}))$; ▷ Trojan branch (2)
  else if $j < i$ and $b_{rev} = 0$ **then**
    Output $\mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m')]; \mathsf{F}_{r'}(\mathsf{lb}))$; ▷ Trojan branch (3)
  else if $j < i$ and $b_{rev} = 1$ **then**
    Output $\mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, \perp]; \mathsf{F}_{r'}(\mathsf{lb}))$; ▷ Trojan branch (4)
  else if $j = i$ **then**
    Output v; ▷ Trojan branch (5)
  end if
end if

  Compute c_{sym}, as random symmetric key ciphertexts of the appropriate size and output $\mathsf{sk}_{\mathsf{lb},f} \leftarrow \mathsf{KeyGen}'(\mathsf{msk}, \mathsf{wrap}[\mathsf{lb}, f, c_{sym}])$.

- $\mathsf{Enc}(\mathsf{mpk}, m, L)$: Choose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 0)$ and compute $y = H_{\mathsf{hk}}(L)$. Compute $c_{pub} \leftarrow \mathsf{Enc}'(\mathsf{mpk}, (\mathsf{hk}, y, r, m, \mathsf{flag} = 0, \mathsf{aux}))$ where $\mathsf{aux} = \perp$ is a "garbage" string of appropriate size. Output $c = (\mathsf{hk}, c_{pub})$.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{lb},f}, L, \mathsf{lb}, c)$: Parse $c = (\mathsf{hk}, c_{pub})$. Run $\mathsf{sk}_{\mathsf{lb},f}$ on c_{pub} to recover an obfuscated program $\widetilde{\mathsf{lock}}$. Compute $\pi \leftarrow \mathsf{Open}(\mathsf{hk}, S, \mathsf{lb})$ and output $z = \widetilde{\mathsf{lock}}(\pi)$.

Parameter sizes. This scheme inherits the parameter sizes from the underlying FE scheme since labels are short and a revocation set L is hashed by the SSB hash. Recently, Ananth and Sahai proposed an adaptively secure FE for Turing machines whose ciphertext, secret key, and master key sizes are optimal [AS15].

Theorem 6.4. *Assuming that $\mathcal{O}, \mathcal{G}, \mathcal{F}, \mathcal{E}$, and $\mathcal{H}$ are secure in the sense of Definition 2.3, 2.4, 2.5, 2.7, and 6.3 respectively, the above scheme is a secure revocable FE in the sense of Definition 6.1.*

Proof. We do a sequence of hybrids, where the initial hybrid game corresponds to the revocable FE security game. We let the set L denote the revoked set chosen by the adversary after mpk is given and let m_0, m_1 denote the two messages chosen by the adversary in the challenge message phase. Let b denote the bit chosen by the the challenger. We use red boxes to highlight changes in hybrid games.

- **Hybrid** 0**:** revocable FE security game.
- **Hybrid** 1**:** Change how secret key queries (lb, f) are answered: initially we choose a key $k \leftarrow \mathsf{SymGen}()$ for the symmetric key encryption scheme. For the j'th secret key query, instead of choosing c_{sym} at random, we compute $\boxed{c_{sym} \leftarrow \mathsf{SymEnc}_k(j, b_{rev}, b_{pre}, w)}$ where:
  - $b_{rev} = 1$ if $\mathsf{lb} \in S$ and $b_{rev} = 0$ otherwise ("revoke bit")
  - If the j'th secret key query occurs before the challenge ciphertext, set $b_{pre} = 1$ ("pre-challenge bit") and set $w = \perp$ to be a garbage string of appropriate size.
  - If the j'th secret key query occurs after the challenge ciphertext, set $b_{pre} = 0$ and $w = \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m_b)]; F_r(\mathsf{lb}))$ where m_0, m_1 are the challenge messages and hk, y, r are the values used to compute the challenge ciphertext.

  Hybrids 0 and 1 are indistinguishable by the pseudorandom ciphertexts security of the symmetric-key encryption.
- **Hybrids** 2**:** In this hybrid, we change how the challenge ciphertext is computed by setting $c_{pub} \leftarrow \mathsf{Enc}'(\mathsf{mpk}, (\mathsf{hk}, y, m = m_b, r, \mathsf{flag}, \mathsf{aux}))$ where $\boxed{\mathsf{flag} = 1}$ and $\boxed{\mathsf{aux} = (i, m', r', k', v)}$ is now selected with $\boxed{i := 0, m' := m_0, r' \text{ a random PRF key}, k' = k}$ matches the key used in hybrid 1 to create the ciphertexts c_{sym}, and $v = \perp$ is a garbage string of appropriate size.

  Hybrids 1 and 2 are indistinguishable by FE security. In this hybrid, the modified ciphertext triggers:
  - Trojan branch (1) of each secret key created post-challenge. In this case, the output is the value w which is hard-coded in the key and exactly matches the value that is output in the real branch.
  - Trojan branch (2) of each secret key created pre-challenge. In this case, the output is the same as in the real branch.
- **Hybrid 3:** Let q_{pre} denotes the number of secret key queries that the adversary makes prior to the challenge message phase. In Hybrid 3 we change how the the encryption is computed to $c_{pub} \leftarrow \mathsf{Enc}'(\mathsf{mpk}, (\mathsf{hk}, y, m = m_b, r, \mathsf{flag} = 1, \mathsf{aux}))$ where we now set $\mathsf{aux} = (i, m', r', k', v)$ with with $\boxed{i := q_{pre} + 1}$, $m' := m_0$, r' a random PRF key, $k' = k$ and $v = \perp$. This is the same as in hybrid 2 except that $i := q_{pre} + 1$. We also choose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, \boxed{\mathsf{lb}_{q_{pre}}})$ to be binding on $\mathsf{lb}_{q_{pre}}$.

  To go from Hybrid 2 to 3 we need to define several intermediate hybrids. For $0 \leq j \leq q_{pre}$ we define Hybrids $2.j.\ell$ with $\ell = 1, \ldots, 6$.

– **Hybrid 2.j.1:** Change the encryption to $c_{pub} \leftarrow \mathsf{Enc}'(\mathsf{mpk}, (\mathsf{hk}, y, m = m_b, r, \mathsf{flag} = 1, \mathsf{aux}))$ where we now set $\mathsf{aux} = (i, m', r', k', v)$ with $i := j$, $m' := m_0$, r' a random PRF key, $k' = k$ and $v = \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}_j, f_j(m_b)]; F_r(\mathsf{lb}_j))$ where (lb_j, f_j) corresponds to the value that was queried in the j'th secret-key query (which was made prior to the challenge ciphertext and is therefore known at this point).

  Hybrid 2.1.1 is indistinguishable from Hybrid 2 by FE security. In hybrid 2.1.1 the ciphertext triggers Trojan branch (1) in all post-challenge keys, and Trojan branch (2) in all pre-challenge keys *except for* the first key ($j = 1$) where it triggers Trojan branch (5). But branches (5) and (2) in key $j = 1$ are identical because of the way that v is chosen.

– **Hybrid 2.j.2:** Change the encryption to $c_{pub} \leftarrow \mathsf{Enc}'(\mathsf{mpk}, (\mathsf{hk}, y, m = m_b, r\{\mathsf{lb}_j\}, \mathsf{flag} = 1, \mathsf{aux}))$ where $\mathsf{aux} = (i, m', r'\{\mathsf{lb}_j\}, k', v)$ and the PRF keys $r\{\mathsf{lb}_j\}$, $r'\{\mathsf{lb}_j\}$ are punctured at lb_j and otherwise everything is the same as in Hybrid 2.j.1. In particular, obfuscated program $v = \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}_j, f_j(m_b)]; F_r(\mathsf{lb}_j))$ is still generated by using pPRF.

  Hybrid $2.j.2$ is indistinguishable from Hybrid $2.j.1$ by correctness of puncturable PRF and FE security.

– **Hybrid** $2.j.3$**:** We now change to choosing $v \leftarrow \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}_j, f_j(m_b)]; U)$ using true randomness U rather than the PRF key. Hereafter, we omit U when we use true randomness to generate obfuscated circuits.

  Hybrid $2.j.3$ is indistinguishable from Hybrid $2.j.2$ by pseudoranomness at punctured points.

– **Hybrid** $2.j.4$**:** Choose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, \mathsf{lb}_j)$.

  Hybrid $2.j.4$ is indistinguishable from Hybrid $2.j.3$ by SSB hash index hiding.

– **Hybrid** $2.j.5$**:** If $\mathsf{lb}_j \in S$ then set $z := \bot$ else $z := f_j(m_0)$. Change $v \leftarrow \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}_j, z])$.

  Hybrid $2.j.5$ is indistinguishable from Hybrid $2.j.6$ by iO security and statistical binding of SSB hash. For $\mathsf{lb}_j \notin L$, we have $f_j(m_0) = f_j(m_1)$. For $\mathsf{lb}_j \in L$, there is no proof π' such that $\mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}_j, z])(\pi')$ outputs z due to the somewhere statistically binding property. Thus, the functional equivalence holds and we can apply iO.

– **Hybrid** $2.j.6$**:** Change to $v = \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}_j, z]; F_{r'}(\mathsf{lb}_j))$.

  Hybrid $2.j.6$ is indistinguishable from Hybrid $2.j.5$ by pseudoranomness at punctured points.

  Hybrid $2.j.6$ is indistinguishable from Hybrid $2.(j+1).1$ for $j < q_{pre}$ by FE security.

  Hybrid $2.q_{pre}.6$ is indistinguishable from Hybrid 3 by FE security.

Together, the above hybrids prove the indistinguishability of hybrids 2 and 3.

- **Hybrid 4:** This is the same as Hybrid 3 but we now set $c_{pub} \leftarrow \mathsf{Enc}'(\mathsf{mpk}, (\mathsf{hk}, y, m = \bot, r = \bot, \mathsf{flag} = 1, \mathsf{aux}))$ where $\mathsf{aux} = (i, m', r', k', v)$ with with $i := q_{pre} + 1$, $m' := m_0$, r' a random PRF key, $k' = k$ and $v = \bot$. In particular, the only difference is that we remove the values m_b and r from the challenge ciphertext. Note that the values m_b, r are still used to answer secret key queries, and in particular to compute the ciphertexts c_{sym}, as defined in Hybrid 1.

  Hybrid 3 and 4 indistinguishable by FE security. In both hybrids, the ciphertext triggers Trojan branch 1 in all post-challenge keys and branches 3,4 in all pre-challenge keys. Either way, the values r, m specified by the challenge ciphertext are never used in these branches.

- **Hybrid 5:** In this hybrid, we change how key queries are answered. In particular, for any secret key query that occurs after the challenge ciphertext we now set $c_{sym} \leftarrow \mathsf{SymEnc}_k(j, b_{rev}, b_{pre}, w)$ with all values chosen as before except that $w \leftarrow \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m_b)])$ is obfuscated using true randomness (rather than using the PRF output $F_r(\mathsf{lb})$ as the randomness for obfuscation).

  Hybrids 4 and 5 are indistinguishable by pseudorandomness at punctured points. Note that the PRF key r does not appear anywhere else in these hybrids other than in computing the values w.

- **Hybrid 6:** In this hybrid, we change how key queries are answered. In particular, for any secret key query that occurs after the challenge ciphertext we now set $c_{sym} \leftarrow \mathsf{SymEnc}_k(j, b_{rev}, b_{pre}, w)$ where $w \leftarrow \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m_0)])$ if $\mathsf{lb} \notin L$ and $w \leftarrow \mathcal{O}(\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, \bot])$ if $\mathsf{lb} \in L$. Note that for the queries where $\mathsf{lb} \notin L$, we have $f(m_0) = f(m_b)$ and therefore these are answered identically in Hybrids 5 and 6, but when $\mathsf{lb} \in L$ the queries are answered differently.

  Let $s = |S|$ denote the number of revoked identities, and let's order them as $\mathsf{lb}_1, \ldots, \mathsf{lb}_s$. To show the indistinguishability of Hybrids 5 and 6 we define intermediate Hybrids 5.j.1, 5.j.2 for $j = 1, \ldots, s$ as follows:

  - **Hybrid** $5.j.1$ In this hybrid, we choose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, \mathsf{lb}_j)$. Furthermore, for any key-query with identity $\mathsf{lb} = \mathsf{lb}_i \in L$,
    * if $i \geq j$ then answer the query as in hybrid 5 and
    * if $i < j$ then answer as in hybrid 6.

    Hybrid 5.j.1 is indistinguishable from Hybrid 5 by SSB hash index hiding.
  - **Hybrid** $5.j.2$ In this hybrid, we choose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, \mathsf{lb}_j)$. Furthermore, for any key-query with identity $\mathsf{lb} = \mathsf{lb}_i \in L$,
    * if $i > j$ then answer the query as in hybrid 5 and
    * if $i \leq j$ then answer as in hybrid 6 (this differs from hybrid $5.j.1$ when $i = j$).

    Hybrid $5.j.2$ is indistinguishable from Hybrid 5.j.1 by iO security and the statistical binding property of the SSB hash. In particular the programs $\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, f(m_b)]$ and $\mathsf{lock}[\mathsf{hk}, y, \mathsf{lb}, \bot]$ are functionally equivalent when hk is binding on lb and $\mathsf{lb} \in L$.

    Hybrid $5.s.2$ is identical to Hybrid 6.

  Together, this shows that Hybrid 5 is indistinguishable from Hybrid 6.

Finally, we notice that in Hybrid 6, the challenge bit b is not used anywhere at all. Therefore, the probability of the adversary guessing b correctly is exactly $\frac{1}{2}$ in Hybrid 6. By the indistinguishability of Hybrids 0 and 6, the probability that the adversary guesses b correctly in the real game is at most negligibly close to $\frac{1}{2}$ as we wanted to show. □

6.4 Trace and Revoke with Flexible Identities from Revocable FE

In this section, we give an outline to construct flexible trace and revoke schemes from revocable FE schemes.

6.4.1 PBE with revocation from Revocable FE

First, we show how to construct PBE with revocation schemes from revocable FE schemes. The definition of PBE with revocation is as follows.

Definition 6.5. 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 and $\mathcal{L}$ a label space. A Private Broadcast Encryption with Revocation 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.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$ takes as input the master secret msk, a (short) label $\mathsf{lb} \in \mathcal{L}$, and a user identity $\mathsf{id} \in \mathcal{ID}$. It outputs a secret key $\mathsf{sk}_{\mathsf{lb},\mathsf{id}}$ for $(\mathsf{lb}, \mathsf{id})$.
- $\mathsf{Enc}(\mathsf{mpk}, S, L, m)$ takes as input the master public key mpk, a *secret* set $S \in \mathcal{S}$, a (public) revocation set L, and a message $m \in \mathcal{M}$. It outputs a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{lb},\mathsf{id}}, L, \mathsf{lb}, c)$ takes as input the secret key $\mathsf{sk}_{\mathsf{lb},\mathsf{id}}$ for a user id whose label lb, a revocation set L, and a ciphertext c. It outputs a message $m \in \mathcal{M}$ or a special symbol $\perp$.
- **Correctness.** For any secret set $S \in \mathcal{S}$, revocation list $L \subseteq \mathcal{L}$, pair of label and identity $(\mathsf{lb}, \mathsf{id})$ such that $(\mathsf{lb} \notin L) \wedge (\mathsf{id} \in S)$, any pair of label and identity $(\mathsf{lb}', \mathsf{id}')$ such that $(\mathsf{lb}' \in L) \vee (\mathsf{id}' \notin S)$, any message $m \in \mathcal{M}$, we have that

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

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

- **Message and Set Hiding.** Intuitively, we ask that for id that are not explicitly allowed to decrypt a ciphertext c or are in a revocation list, 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 $\mathcal{A}$.
  - The attacker $\mathcal{A}$ specifies a "revoke set" $L^* \subseteq \mathcal{L}$.
  - $\mathcal{A}$ is allowed to make arbitrary keygen queries, where it sends a pair of label and identity $(\mathsf{lb}, \mathsf{id}) \in \mathcal{L} \times \mathcal{ID}$ to the challenger, and the challenger responds with $\mathsf{sk}_{\mathsf{lb},\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$. The challenger also records $(\mathsf{lb}, \mathsf{id})$ in a list $\mathcal{R}$.
  - At some point, $\mathcal{A}$ makes a single challenge query, where is 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^*: $c^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, S_b^*, L^*, m_b^*)$. 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, for any $(\mathsf{lb}, \mathsf{id}) \in \mathcal{R}$, $(\mathsf{id} \notin S_0^*) \wedge (\mathsf{id} \notin S_1^*)$ or $\mathsf{lb} \in L^*$.

∗ 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, for any $(\mathsf{lb}, \mathsf{id}) \in \mathcal{R}$, $\mathsf{id} \notin S_0^* \Delta S_1^*$ or $\mathsf{lb} \in L^*$, 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 pairs of label and identity $(\mathsf{lb}, \mathsf{id})$, subject to the constraint that $(\mathsf{lb}, \mathsf{id})$ must satisfy the same checks as above: if $m_0^* \neq m_1^*$, then $(\mathsf{id} \notin S_0^*) \wedge (\mathsf{id} \notin S_1^*)$ or $\mathsf{lb} \in L^*$, and if $S_0^* \neq S_1^*$, then $\mathsf{id} \notin S_0^* \Delta S_1^*$ or $\mathsf{lb} \in L^*$. If the adversary tries to query in a $(\mathsf{lb}, \mathsf{id})$ that fails the check, the challenger outputs a random bit and aborts the game.

– Finally, $\mathcal{A}$ 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 encryption with revocation is secure if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.

In our revocable FE security game, a challenge revocation set is determined before secret queries, but a challenge message is adaptively chosen. This means a private set S in PBE is *adaptively* chosen in the security game of PBE. Thus, we can construct adaptively secure PBE schemes.

Construction. Let $(\mathsf{revFE.Setup}, \mathsf{revFE.KeyGen}, \mathsf{revFE.Enc}, \mathsf{revFE.Dec})$ be a secure revocable functional encryption scheme for label space $\mathcal{L}$, function set $\mathcal{F}$ is the set of function f_{id} that is defined in the description of the algorithm KeyGen, and message space $\mathcal{M}' = \mathcal{S} \times \mathcal{M}$ where $\mathcal{S}$ is a collection of subsets of $\mathcal{ID}$ in PBE with revocation and $\mathcal{M}$ is the plaintext space of PBE with revocation.

The $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}$, and Dec algorithms are as follows:

$\mathsf{Setup}()$**:** This is the same as $\mathsf{revFE.Setup}()$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$**:** To generate a secret key for $(\mathsf{lb}, \mathsf{id})$, run $\mathsf{revFE.KeyGen}(\mathsf{msk}, \mathsf{lb}, f_{\mathsf{id}})$ where

$$f_{\mathsf{id}}(S, m) = \begin{cases} m & \text{if } \mathsf{id} \in S \\ \perp & \text{if } \mathsf{id} \notin S \end{cases}$$

$\mathsf{Enc}(\mathsf{mpk}, S, L, m)$**:** To encrypt a message m with a secret set S and a revocation list L, run $\mathsf{revFE.Enc}(\mathsf{mpk}, m' = (S, m), L)$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{lb},\mathsf{id}}, L, \mathsf{lb}, c)$**:** To decrypt a ciphertext c, run $\mathsf{revFE.Dec}(\mathsf{sk}_{\mathsf{lb},\mathsf{id}}, L, \mathsf{lb}, c)$.

This scheme inherits the parameter sizes from the underlying revocable FE scheme.

Theorem 6.6. *If* $(\mathsf{revFE.Setup}, \mathsf{revFE.KeyGen}, \mathsf{revFE.Enc}, \mathsf{revFE.Dec})$ *is a secure revocable FE scheme in the sense of Definition 6.1, then* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a secure PBE with revocation scheme in the sense of Definition 6.5.*

This holds since revocable FE is a generalization of PBE with revocation.

6.4.2 Trace and Revoke with Flexible Identities from PBE with Revocation

We show how to construct flexible trace and revoke schemes from PBE with revocation schemes. The definition of flexible trace and revoke is as follows.

Definition 6.7. Let $\mathcal{ID}$ be some collection of identities, $\mathcal{L}$ a label space, and $\mathcal{M}$ a message space. A flexible trace and revoke scheme for $\mathcal{M}, \mathcal{L}, \mathcal{ID}$ is a tuple of 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.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$ takes as input the master secret msk, a label $\mathsf{lb} \in \mathcal{L}$, and an identity $\mathsf{id} \in \mathcal{ID}$, and outputs a secret key $\mathsf{sk}_{\mathsf{lb},\mathsf{id}}$ for $(\mathsf{lb}, \mathsf{id})$.
- $\mathsf{Enc}(\mathsf{mpk}, S, L, m)$ takes as input the master public key mpk, a broadcast set S, a revocation list L, and a message $m \in \mathcal{M}$, and outputs a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{lb},\mathsf{id}}, L, \mathsf{lb}, c)$ takes as input the secret key $\mathsf{sk}_{\mathsf{lb},\mathsf{id}}$ for a label lb, an identity id, a revocation list L, a label lb, 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 a parameter q, ϵ, and has oracle access to a decoder algorithm $\mathcal{D}$. It produces an identity $\mathsf{id} \in \mathcal{ID}$, or outputs $\perp$.
- **Correctness.** For any message $m \in \mathcal{M}$, list $S \subseteq \mathcal{ID}$ and $L \subseteq \mathcal{L}$, and pair of label and identity $(\mathsf{lb}, \mathsf{id}) \in \mathcal{L} \times \mathcal{ID}$ and $(\mathsf{lb}', \mathsf{id}') \in \mathcal{L} \times \mathcal{ID}$ such that $(\mathsf{id} \in S \wedge \mathsf{lb} \notin L)$ and $(\mathsf{id} \notin S \vee \mathsf{lb}' \in L)$, we have that

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

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

- **Semantic security.** Informally, we ask that an adversary that does not hold any secret keys that are *not in a revocation list* cannot learn the plaintext m. This is formalized by the following experiment between an adversary $\mathcal{A}$ and challenger:

  1. The challenger runs $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$, and gives mpk to $\mathcal{A}$.
  2. The attacker $\mathcal{A}$ specifies a "revoke set" $L^* \subseteq \mathcal{L}$.
  3. $\mathcal{A}$ is allowed to make key queries, where it sends a pair $(\mathsf{lb}, \mathsf{id}) \in \mathcal{L} \times \mathcal{ID}$ to the challenger, and the challenger responds with $\mathsf{sk}_{\mathsf{lb},\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$. The challenger also records the pair $(\mathsf{lb}, \mathsf{id})$ in a list $\mathcal{R}$.
  4. $\mathcal{A}$ makes a challenge query where it submits a set S^* and two messages m_0^*, m_1^* such that for any $(\mathsf{lb}, \mathsf{id}) \in \mathcal{R}$, $\mathsf{id} \notin S^*$ or $\mathsf{lb} \in L^*$. The challenger chooses a random bit b, and responds with the encryption of m_b^*: $c^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, S^*, L^*, m_b^*)$.
  5. Again, $\mathcal{A}$ is allowed to make key queries under the condition above.
  6. $\mathcal{A}$ 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. We say 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 is sends a pair of label and identity $(\mathsf{lb}, \mathsf{id}) \in \mathcal{L} \times \mathcal{ID}$ to the challenger, and the challenger responds with $\mathsf{sk}_{\mathsf{lb},\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$. The challenger also records the identities queries in a list $\mathcal{R}$ (In this case, no need record labels).
  - $\mathcal{A}$ 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, $q = |\mathcal{R}|$). 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 $\mathcal{R}$. That is, $\mathcal{T} \setminus \mathcal{R} \neq \emptyset$ or
    * Both of the following hold:
      · $\mathcal{D}$ is ϵ-useful, meaning $\Pr[\mathcal{D}(c) = m_b^* : b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{mpk}, S = \mathcal{ID}, L = \emptyset, m_b^*)] \geq \frac{1}{2} + \epsilon$[5].
      · $\mathcal{T}$ does not contain at least one user inside $\mathcal{R}$. That is, $\mathcal{T} \cap \mathcal{R} = \emptyset$.

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

  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.

Again, a secret set S is *adaptively* chosen by adversaries in the security game of PBE and the flexible trace and revoke scheme is adaptively secure.

Construction. Let $(\mathsf{PBE.Setup}, \mathsf{PBE.KeyGen}, \mathsf{PBE.Enc}, \mathsf{PBE.Dec})$ be a secure private broadcast encryption with revocation scheme for label space $\mathcal{L} = [Q]$ where Q is a polynomial in λ, identity space $\mathcal{ID} = [r] \times [2^{\lambda+1}]$. We construct a flexible trace and revoke scheme for an identity space $\mathcal{ID}' = \{0,1\}^r$.

$\mathsf{Setup}()$**:** This is the same as $\mathsf{PBE.Setup}()$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id}')$**:** To generate the secret key for an identity $\mathsf{id}' \in \mathcal{ID}'$, write $\mathsf{id}' = (\mathsf{id}'_1, \ldots, \mathsf{id}'_r)$ where $\mathsf{id}'_i \in \{0,1\}$. Choose a random $s \in [2^\lambda]$, and define the identity $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_r) \in \mathcal{ID}$ where $\mathsf{id}_i = 2s - \mathsf{id}'_i \in \mathcal{ID}_0$. Run $\mathsf{PBE.KeyGen}(\mathsf{msk}, \mathsf{lb}, \mathsf{id})$, and output the resulting secret key.

[5]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 produce. 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.

$\mathsf{Enc}(\mathsf{mpk}, S, L, m)$**:** Run $\mathsf{PBE.Enc}(\mathsf{mpk}, S, L, m)$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{lb},\mathsf{id}}, L, \mathsf{lb}, c)$**:** Run $\mathsf{PBE.Dec}(\mathsf{sk}_{\mathsf{lb},\mathsf{id}}, L, \mathsf{lb}, c)$.

$\mathsf{Trace}^{\mathcal{D}}(\mathsf{mpk}, m_0, m_1, q, \epsilon)$**:** We can similarly construct this algorithm using QTrace' as that in Section 4 and 5 except that, for generation of ciphertexts, we run $\mathsf{Enc}(\mathsf{mpk}, S = \mathcal{ID}, L = \emptyset, m_b)$.

This scheme inherits the parameter sizes from the underlying PBE with revocation.

Theorem 6.8. *If* $(\mathsf{PBE.Setup}, \mathsf{PBE.KeyGen}, \mathsf{PBE.Enc}, \mathsf{PBE.Dec})$ *is a secure PBE with revocation scheme in the sense of Definition 6.5 for identity space* $\mathcal{ID}$ *and private class* $\mathcal{S}$*, where* $\mathcal{ID}$ *is defined as above, then there is an efficient algorithm* Trace *such that* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ *as defined above is a secure flexible trace and revoke scheme in the sense of Definition 6.7.*

We can prove this in a similar way as that of flexible traitor tracing from PBE in Section 5.

Parameter sizes. The trace and revoke system inherits the parameter sizes from the underlying FE scheme. (Note that the ciphertext size of our revocable FE is independent of the size of revocation lists.) Therefore, using a compact functional encryption scheme [AS15], we obtain a trace and revoke system where ciphertexts have size $|m| + \mathsf{poly}(\lambda)$ and secret keys have size $\mathsf{poly}(\lambda, r)$, which is optimal as explained in Section 5.

Remark 6.9. Someone may think revocation lists are redundant to achieve trace and revoke schemes since we can revoke users by eliminating users identities from the set S. However, in some settings (e.g., anonymous setting), we would like to hide identities that includes sensitive information and reveal only short random labels. When such secret identities are unknown, then we cannot revoke users by changing a set S. We can revoke users by using a revoke set L that includes only *public* labels.

References

[ABSV14] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. Cryptology ePrint Archive, Report 2014/917, 2014. http://eprint.iacr.org/2014/917.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. Cryptology ePrint Archive, Report 2015/173, 2015. http://eprint.iacr.org/2015/173.

[AS15] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines, 2015. Unpublished manuscript.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 52–73, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany.

[BF99] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Berlin, Germany.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 573–592, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. Cryptology ePrint Archive, Report 2015/163, 2015. http://eprint.iacr.org/2015/163.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 211–220, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Berlin, Germany.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Berlin, Germany.

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

[CHV15] Aloni Cohen, Justin Holmgren, and Vinod Vaikuntanathan. Publicly verifiable software watermarking. Cryptology ePrint Archive, Report 2015/373, 2015. http://eprint.iacr.org/2015/373.

[CPP05] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 542–558, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.

[DF02] Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 61–80. Springer, 2002.

[DF03] Yevgeniy Dodis and Nelly Fazio. Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 100–115, Miami, USA, January 6–8, 2003. Springer, Berlin, Germany.

[DFKY05] Yevgeniy Dodis, Nelly Fazio, Aggelos Kiayias, and Moti Yung. Scalable public-key tracing and revoking. *Distributed Computing*, 17(4):323–347, 2005.

[FT99] Amos Fiat and Tamir Tassa. Dynamic traitor training. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 354–371, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014. http://eprint.iacr.org/2014/666.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GST04] Michael T. Goodrich, Jonathan Z. Sun, and Roberto Tamassia. Efficient tree-based revocation in groups of low-state devices. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 511–527, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Berlin, Germany.

[GSY99] Eli Gafni, Jessica Staddon, and Yiqun Lisa Yin. Efficient methods for integrating traceability and broadcast encryption. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*,

volume 1666 of *Lecture Notes in Computer Science*, pages 372–387, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany.

[HS02] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Berlin, Germany.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery.

[KD98] Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 145–157, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[KY02] Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 450–465, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[NP98] Moni Naor and Benny Pinkas. Threshold traitor tracing. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 502–517, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Berlin, Germany.

[NP01] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In Yair Frankel, editor, *FC 2000: 4th International Conference on Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 1–20, Anguilla, British West Indies, February 20–24, 2001. Springer, Berlin, Germany.

[NW15] Ryo Nishimaki and Daniel Wichs. Watermarking cryptographic programs against arbitrary removal strategies. Cryptology ePrint Archive, Report 2015/344, 2015. http://eprint.iacr.org/2015/344.

[SSW01] Alice Silverberg, Jessica Staddon, and Judy L. Walker. Efficient traitor tracing algorithms using list decoding. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 175–192, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Germany.

[SW00] Reihaneh Safavi-Naini and Yejing Wang. Sequential traitor tracing. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 316–332, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Berlin, Germany.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[TT01] Wen-Guey Tzeng and Zhi-Jia Tzeng. A public-key traitor tracing scheme with revocation using dynamic shares. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 207–224, Cheju Island, South Korea, February 13–15, 2001. Springer, Berlin, Germany.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014. http://eprint.iacr.org/2014/588.

[Zha14] Mark Zhandry. Adaptively secure broadcast encryption with small system parameters. Cryptology ePrint Archive, Report 2014/757, 2014. http://eprint.iacr.org/2014/757.