White Box Traitor Tracing

Mark Zhandry
Princeton University & NTT Research, USA

Abstract

Traitor tracing aims to identify the source of leaked decryption keys. Since the "traitor" can try to hide their key within obfuscated code in order to evade tracing, the tracing algorithm should work for general, potentially obfuscated, decoder *programs*. In the setting of such general decoder programs, prior work uses *black box* tracing: the tracing algorithm ignores the implementation of the decoder, and instead traces just by making queries to the decoder and observing the outputs.

We observe that, in some settings, such black box tracing leads to consistency and user privacy issues. On the other hand, these issues do not appear inherent to *white box* tracing, where the tracing algorithm actually inspects the decoder implementation. We therefore develop new white box traitor tracing schemes providing consistency and/or privacy. Our schemes can be instantiated under various assumptions ranging from public key encryption and NIZKs to indistinguishability obfuscation, with different trade-offs. To the best of our knowledge, ours is the first work to consider white box tracing in the general decoder setting.

1 Introduction

Traitor tracing [CFN94] deters piracy by embedding identifying information into users' personalized decryption keys. From a leaked key, it should be possible to "trace," extracting the "traitor's" identifying information; with this information, remedial action can be taken such as fines, prosecution, and/or revocation. Tracing is ideally possible even in a variety of adversarial scenarios, such as if many users collude or if the secret key is hidden inside an obfuscated decoder program. The bulk of the tracing literature has focused on reducing the sizes of various components, such as ciphertexts and public and secret keys.

Analyzing (potentially obfuscated) program code is notoriously difficult. Consequently, most recent traitor tracing works (e.g. [CFN94, BSW06, BN08, BZ14, NWZ16, GKW18, Zha20]) operate in a *black box* model: the tracer actually does *not* try to inspect the particular software of the decoder, but instead simply queries the decoder on various ciphertexts and observes the outputs. From just the input/output behavior, the tracer is able to extract the identifying information[1].

This work: white box tracing. In this work, we consider the use of *white box* algorithms for tracing general decoders. Specifically, we allow the adversary to produce arbitrary (potentially obfuscated) programs in an attempt to remove the embedded identifying information, but use non-black box algorithms for tracing. The two main questions we explore in this work are:

[1] Another oft-cited reason to consider black box tracing is that the decoder could be contained in a hardware device employing various tamper resistant mechanisms to prevent it's code from being inspected. In this work, however, we will only consider software decoders.

What are potential advantages of white box tracing? And,
How to white box trace general adversarial decoders?

Remark 1. An early model for traitor tracing, which we will call *faked key* tracing, stipulates that the traitor outputs an actual valid key for the system. Tracing then uses the combinatorial or algebraic structure of the key, as opposed to its input/output behavior. Many early traitor tracing works consider faked key tracing [KD98, BF99, NP01, KY03, TS06, JKL09, JKL09, ADVW13], and some refer to this model as "white box tracing" or "non-black box tracing" (see, e.g. [NDC+15] and [GNPT13], respectively). Such naming reflects that non-black box tracing algorithms have always coincided with tracing models where the traitor must output a valid key. Outside of traitor tracing, however, the labels "white box" or "non-black box" refer to the type of access to a program, and is potentially orthogonal to the format of the program. We therefore prefer the terms "faked key" versus "general decoder" to refer to the structure of the adversary's decoder, and terms "black box" versus "white box" to refer to the level of access the tracing algorithm has to the decoder. To the best of our knowledge, ours is the first work exploring white box tracing in the general decoder setting.

1.1 Motivation

To motivate white box tracing, we now discuss limitations of black box tracing; overcoming these limitations will be the focus of our work. These limitations are orthogonal to the "usual" goal of traitor tracing, namely minimizing parameter sizes. As such, parameter sizes are only a secondary consideration in this work.

Public tracing. We first motivate a particular type of traitor tracing which has both *public tracing* and *embedded identities*. Embedded identities, originally proposed by Nishimaki et al. [NWZ16], means that arbitrary information can be embedded in the secret keys; in contrast, most tracing schemes only embed an index from a polynomial-sized set. Nishimaki et al. point out that the tracer would naturally want to know useful identifying information about the traitor, in order to prosecute or fine. The key issuer could of course maintain a database mapping user indices to actual identifying information, but having to store such a database in the clear naturally creates privacy concerns. Embedded identities allow this information to be stored directly in the issued keys themselves, eschewing the need for such a database.

For public tracing, the tracing algorithm only needs the public key and no secrets. This is in contrast to *secret tracing*, where a secret key is required to trace, and anyone with the secret key can break the security of the system. There are at least a few reasons to prefer public tracing algorithms:

- Secret key tracing means the tracer cannot be compromised. Public key tracing allows *anyone* to trace, removing a potential point of failure.
- As explained in [Pfi96], private tracing provides no natural mechanism for submitting evidence to a judge, as there is no way besides revealing the secret tracing key to certify the results of tracing. While there are solutions in the secret tracing setting, public tracing automatically solves the problem: the judge can always verify by simply re-tracing with the public key.
- In private key tracing, the tracer must somehow discover the decoder in order to trace, and deterrence therefore relies on such discovery. It may take time for the tracer to discover the

decoder program, or it may never be discovered if the traitor and an unauthorized user are secretive enough in their communication. After all, the pirate decoder would naturally be transmitted out of band, and there is no reason to believe the tracer would automatically see the decoder.

In contrast, with public key tracing, the sensitive information is immediately revealed to *anyone* who receives the decoder, including the un-authorized user. Especially when combined with embedded identities, public tracing yields a very strong deterrent mechanism: for example, if the embedded information contains a bank account number, then a traitor would likely be unwilling to send their key to anyone else, especially the unscrupulous un-authorized user.

The good news is that there already exist such public embedded identity tracing schemes, with different trade-offs in terms of parameter sizes and assumptions used [NWZ16, GKW19]. However, as we will now explain, public *black box* tracing schemes, including all existing public tracing schemes for general decoders, are inherently vulnerable to certain kinds of attacks.

Problem 1: Privacy. Consider an encrypted group chat application, where a group of users broadcast messages to the entire group. The broadcasts are encrypted to protect against eavesdropping. The group members are also mutually distrusting, and want to protect against a traitor revealing their key to an outside user. For example, the group could consist of political dissidents coordinating a protest against an authoritarian regime, and they are concerned that a member may give their key to government agents.

The group therefore will use a traitor tracing scheme to encrypt their message, embedding sensitive or identifying information of each user into personalized decryption keys. In this decentralized scenario, it is unclear who the tracer should be, and also unclear how to securely maintain a database of users' identifying information. Therefore, the group would naturally want a scheme with public tracing and embedded identities, as discussed above[2].

Unfortunately, we observe that such constructions inherently come with privacy concerns, due to having black box tracing. A malicious Alice (whether or not an authorized group member) may try to steal Bob's private information by running the tracing algorithm over the network. That is, Alice can send messages to Bob, and see how he responds, mounting a chosen ciphertext attack against Bob's decryption functionality. This is exactly how black box tracing works, and since the scheme has public tracing, Alice has all the access she needs to trace Bob's key. In fact, tracing algorithms typically work in the "minimal access" setting, meaning Alice only needs to know whether Bob decrypts. In our political dissident scenario above, this means government agents may be able to learn the identities of the dissidents (or whatever sensitive information is embedded in the user keys) by simply posting messages to the group chat, and seeing if there are any responses. This would naturally concern the group members.

Thus, it is impossible to get the best of all worlds—public tracing, embedded sensitive information, and user privacy under chosen ciphertext attacks—with black box tracing. White box tracing, on the other hand, may offer a solution: the remote attacker never actually sees the user's decryption program, and may therefore be unable to run a white box tracing algorithm. Of course, at this point it is not clear how to actually use white box tracing to achieve these goals.

[2]Similar to [NWZ16], we envision the original setup and key distribution executed through multiparty computation, so no single user is responsible for setup.

Remark 2. Nishimaki et al. consider an object they call "anonymous" traitor tracing, where the users never reveal their identifying information to the key issuer. However, beyond motivating the direct embedding of sensitive information within user keys, they do not further explore anonymity or privacy in traitor tracing. In particular, they do not discuss the privacy issue we observe here.

Problem 2: Consistency. In traitor tracing, the functionality of the various user keys is different. This is inherent, for similar reasons to the case of watermarking as explained by Barak et al. [BGI+01]: if each user key had identical functionality, then a traitor could apply *indistinguishability obfuscation* (iO) to their key. The guarantees of iO would then imply the obfuscations of different users' keys are computationally indistinguishable, and hence cannot be efficiently traced. Thus any efficient tracing scheme that maintains perfect consistency would necessarily prove the non-existence of iO, which currently seems out of reach.

This means certain ciphertexts will decrypt differently under different user keys. While these differing inputs would not occur under normal operation, it is nonetheless easy to find differing inputs in the case of *public* black box tracing: since the tracing algorithm must distinguish between the keys by querying the decoder functionality, it must be querying exactly on these differing inputs.

To see why this might be an issue, consider executing a multi-party computation (MPC) protocol. As is standard in the MPC literature, assume that the users have access to a reliable broadcast channel: when one user sends a message on the channel, all other users are guaranteed to receive the same message. Now, suppose that the set of users want to encrypt the broadcasts in their MPC protocol using a traitor tracing scheme[3]. Unfortunately, even if the ciphertexts are sent over a reliable broadcast channel, the MPC is run on a virtual plaintext channel that is *not* reliable: a malicious user may broadcast a ciphertext specifically designed to decrypt differently by different users. Typical MPC protocols require broadcasts to be received consistently between the various users, and such an inconsistent decryption would break the guarantees of the MPC protocol[4].

More generally, in any setting where a broadcast channel is needed to ensure that all users receive the same message, encrypting the communication with a public black box tracing scheme can result in an unreliable broadcast channel.

On the other hand, such consistency issues do not appear inherent to white box tracing: a malicious user only has black box access to the other users' decryption functionalities, and may be unable to use this access to find an input that decrypts differently. Yet if a user actually leaks their decryption key or an obfuscated decoder with the key inside, a white box tracing algorithm might nevertheless be able to trace, and in particular may be able to find such a differing ciphertext by inspecting the code. Again, it is not yet clear how exactly to use white box tracing to achieve this goal.

1.2 Overview of Our Results

In this work, we give several new results for white box traitor tracing:

- In Section 3, we give definitions for privacy and consistency in the traitor tracing setting. Formalizing the above observations, we show that it is impossible to satisfy either privacy

[3] Of course, MPC security already implies that outsiders will be unable to learn anything about the users' inputs even without encrypting. But perhaps the users are encrypting messages for other reasons.

[4] There are MPC protocols that do not need broadcast channels, but they often come at the cost of increased round complexity (see [CGZ20] and references therein).

or consistency while simultaneously achieving the tracing guarantee with a public black box tracing algorithm.

- In Section 5, we construct a secure white box public tracing system that also satisfies our privacy notion. Our construction can be based on either generic public key encryption and non-interactive zero knowledge, or on indistinguishability obfuscation (iO), with different trade-offs in terms of collusion resistance and parameter sizes. This scheme is not consistent.

- We do not fully solve the consistency problem, but in Section 6 we demonstrate a white box traitor tracing scheme that achieves the tracing guarantee for *constant*-sized collusions, while also achieving consistency and privacy (both for arbitrary collusions). Our scheme uses fully homomorphic encryption, compute-and-compare obfuscation, and non-interactive zero knowledge, which are all implied by circularly secure Learning With Errors (LWE) or iO.

Along the way, we introduce and build a notion of black box function privacy for functional encryption (Section 4), which may be of independent interest.

1.3 Future Directions

Our work motivates several fascinating future directions:

- We are able to construct consistent tracing only for constant-sized collusions. Is arbitrary-collusion consistent tracing possible, potentially even under extremely strong assumptions? Alternatively, is there an impossibility?

- Our constructions utilize heavy machinery, using non-black box techniques at many levels. This leads our constructions to be inefficient. Can truly efficient white box tracing be achieved?

- Traitor tracing can be seen as a special case of the more general problem of software watermarking. To the best of our knowledge, all works in the watermarking setting also use black box mark detection/extraction, and the privacy and consistency issues naturally translate to watermarking. Can white box techniques be used to overcome similar issues in watermarking?

- There is a large gap between known programs that can be watermarked (e.g. puncturable PRFs [CHN+16]) and programs that are known to be un-watermarkable (e.g. un-obfuscatable functions [BGI+01]). Can white box mark detection/extraction be use to help close this gap?

- Chosen ciphertext attacks for traitor tracing has been considered before (e.g. [DF03]), but to the best of our knowledge, prior such explorations have been limited to message secrecy goals. Our work highlights further consequences of CCA attacks. Are there other possible consequences?

2 Our Techniques

2.1 Part 1: Private Traitor Tracing

Definitions and impossibility. In Section 3.2, we give a formal model for privacy of a user's sensitive information under chosen ciphertext queries to their decryption functionality. We give several indistinguishability and simulation-based definitions formalizing "learning nothing"; we show

that the indistinguishability notions are equivalent to the corresponding simulation-based notions. Our strongest notion actually gives the adversary the full master secret key, meaning privacy holds even if the master key is leaked. We also formalize the above observations, showing that even the weakest versions of our privacy notion are impossible for black box public tracing schemes. Our strongest notion of privacy (where the adversary gets the master secret key) is even incompatible with black box *secret* tracing.

Theorem 1 (Informal)**.** *Black box publicly traceable schemes cannot be private.*

Achieving privacy through white box tracing. In Section 5 turn to building a scheme that can be publicly traced while maintaining privacy. Since black box tracing is impossible, we must devise a tracing scheme that is inherently white box, in that accessing the code of decoder allows for tracing while black box access cannot.

The natural starting point are the un-obfuscatable functions (UOFs) of Barak et al. [BGI+01], which can be learned from *any* code for the function, but not from black box access. While UOFs are indeed closely related to our goal, they do not immediately give what we need:

- UOFs are not necessarily decryption functions. While Barak et al. show how to extend their UOFs to *encryption* functions, it is not obvious that they can be extended to the *decryption* functionality.
- In Barak et al.'s UOF, the functionality of the encryption scheme is tied to the UOF itself. In the traitor tracing system, we want many different users to be able to decrypt the same ciphertext, hence seemingly all keys would have the same UOF. But at the same time, the users should have different sensitive information, seemingly requiring different UOFs.
- Barak et al.'s UOF does not handle the case of colluding users.
- Barak et al.'s UOF requires perfect or near-perfect correctness for the decoder. While this can be extended to much lower correctness using *robust* un-obfuscatable function [BP13], the current techniques do not extend to the inverse-polynomial correctness setting, as usually required in traitor tracing.

Our idea is to use *black box* traitor tracing techniques, but set the embedded information for a user to be a UOF in order to upgrade the black box scheme into a white box scheme. However, this requires care. The naive approach is to set the embedded information to be the actual code of the UOF, but this will not work: the adversary can mount the black box tracing algorithm remotely to recover the UOF code, and then recover the user's private information from the UOF code. We therefore need a more sophisticated embedding.

To describe our solution, we first recall the black box tracing scheme of Nishimaki et al. [NWZ16], which follows the PLBE framework [BSW06]. Start from a *public key functional encryption* (FE), which allows for generating secret keys sk_f for functions g; sk_f allows for learning $f(x)$ from an encryption of x, but nothing else about x. Assume the identity space is $[I]$, for some exponentially-large integer I. Nishimaki et al. encrypt a message m by FE-encrypting the pair $(0, m)$. The secret key with identity id embedded is then $\mathsf{sk}_{f_{\mathsf{id}}}$ where

$$f_{\mathsf{id}}(z, m) = \begin{cases} m & \text{if } \mathsf{id} > z \\ \bot & \text{if } \mathsf{id} \leq z \end{cases} .$$

Notice that $f_{\mathsf{id}}(0, m) = m$, meaning $\mathsf{sk}_{f_{\mathsf{id}}}$ will decrypt honest ciphertexts, while $f_{\mathsf{id}}(I, m) = \perp$. By FE security, given any decoder D built from $\mathsf{sk}_{f_{\mathsf{id}}}$ and any z, one can test if $\mathsf{id} > z$ by looking at the decoder's decryption probability on encryptions of (z, m). A binary search over z can then recover id; Nishimaki et al. show how to extend the binary search to the case of colluding users and to decoders with small decryption probability, as required for traitor tracing.

In order to embed a *function* into the secret key, rather than just a string, we modify Nishimaki et al.'s construction as follows. To embed a function g, we choose a random "tag" τ, and generate the function secret key $\mathsf{sk}_{f_{g,\tau}}$ where

$$f_{g,\tau}(z, x, m) = \begin{cases} m & \text{if } \tau > z \\ \perp & \text{if } (\tau < z) \vee (\tau = z \wedge x = \perp) \\ m & \text{if } \tau = z \wedge x \neq \perp \wedge g(x) = 1 \\ \perp & \text{if } \tau = z \wedge x \neq \perp \wedge g(x) = 0 \end{cases}.$$

If we set $x = \perp$, then $\mathsf{sk}_{f_{g,\tau}}$ has the same structure as f_{id} above with $\mathsf{id} = \tau$. Therefore, we can first run a binary over z search to recover τ. Then we evaluate g on an input x by testing the decoder on encryptions of (τ, x, m), and seeing whether it is able to decrypt; if it can decrypt, we must have $g(x) = 1$, otherwise $g(x) = 0$. The result is what we call *function-embedded* traitor tracing (FETT).

Remark 3. The structure above is similar to an optimization Nishimaki et al. employ to get a scheme where ciphertexts are very short, in particular shorter than the identity id. Essentially, they let g be the function with polynomial-sized domain whose truth table is id. The structure was also used in [GKW19], again with g being a truth table, with the goal of achieving efficient traitor tracing under standard assumptions. Our use of this structure is with an entirely different goal: to embedding functions in a non-trivial way into the secret keys.

With a FETT, we can now build our private traitor tracing scheme by setting g to be an un-obfuscatable function $\mathsf{UOF}_{\mathsf{id}}$, which has the identity id of the user embedded. Given a decoder D, we can construct a program P that evaluates $\mathsf{UOF}_{\mathsf{id}}$ by running the FETT tracing algorithm as described above. The un-obfuscatable function guarantee means that from P, we can extract the identity id. Meanwhile, the intuition for privacy is that a remote user can only make black box queries to the user, corresponding to black box queries to $\mathsf{UOF}_{\mathsf{id}}$; the un-obfuscatable function guarantee means that such queries do not reveal id. Realizing this intuition, however, comes with several challenges:

- Since tracing is randomized, the program P is a randomized procedure, whereas the un-obfuscatable function guarantee of Barak et al. only applied to deterministic circuits. One can make P deterministic by hard-coding the randomness, but for any particular choice of randomness there may be some x where P outputs the incorrect answer. Additionally, the original P may actually completely fail to evaluate $\mathsf{UOF}_{\mathsf{id}}$ correctly on some inputs, for example if x is the master secret key. Fortunately, we show that P correctly computes $\mathsf{UOF}_{\mathsf{id}}$ for inputs x that are efficiently computable to the adversary. We show that the Barak et al. functions maintain the un-obfuscatable function guarantee even for this relaxed notion of correctness for P [5].

[5] Our needed notion is also implied by *robust* UOFs [BP13], but these are only known from trapdoor permutations. Barak et al.'s construction relies on just one-way functions, which are implied by FE.

- For a secure FE, black box queries to the secret key sk_f might still reveal the code for f. If such an FE is used in our construction, the result is that a remote adversary may be able to obtain the code for $\mathsf{UOF}_{\mathsf{id}}$, and hence id. As such, we actually need a function privacy notion for the FE scheme, which roughly requires that black box queries to sk_f are "no better than" black box queries to f itself. This notion of function privacy is incompatible with existing notions of privacy for public key functional encryption[6]. In Section 4, we show a simple transformation upgrading any plain FE scheme into one with our notion of function privacy using non-interactive zero knowledge (NIZK) proofs.

Remark 4. Our "black box" function privacy notion may have applications beyond this work. For example, consider a common-cited application of functional encryption to filtering spam. Here, f is a spam filter employed by an email server. A user wants the server to be able to route encrypted emails according to the spam filter, but does not want the server to learn anything beyond whether or not an email was spam. The solution is to give the server sk_f. But now suppose that f is proprietary, and the server wants to prevent potential spammers from learning too much about f [7]. A spammer can effectively query f by sending spam to the user and seeing whether or not the user actually receives the email (as indicated, say, by whether the user clicks on a link). Plain functional encryption, unfortunately, may allow the adversary to do more: the result of decrypting malicious ciphertexts may reveal the bits of sk_f, or even the code of f. Black box function privacy guarantees that the spammer is limited to just querying f and learning the input/output behavior of f.

Instantiations. We can plug any FE scheme (and NIZK) into our construction. Our conversions preserve the ciphertext sizes of the underlying FE. Using known FE constructions, we obtain the following:

Theorem 2 (Informal)**.** *Assuming public key encryption and NIZKs, there exists a traitor tracing scheme with public tracing, embedded identities, and privacy, with* $\mathsf{poly}(N)$*-sized ciphertexts for collusions of size* N*. Assuming indistinguishability obfuscation and one-way functions, there exists such a scheme with* $O(1)$*-sized ciphertexts.*

2.2 Part 2: Toward Consistent Traitor Tracing

Next, we turn to the problem of making sure different users decrypt consistently.

Definition and impossibility. In Section 3.3, we define several variants of consistency. The strongest requires that even if the master secret key is leaked, it is impossible to find a ciphertext that decrypts differently under any two users. This variant is quite strong, and we do not know how to achieve it. Instead we also consider weaker variants. The variant we ultimately achieve requires that a malicious user, or group of users, cannot find a ciphertext that would cause two *honest* users to decrypt differently. Note that our notion *does* allow a group of malicious users to find ciphertexts that *they* decrypt differently (or that they decrypt differently than the honest user(s)). We nevertheless believe our notion is meaningful, as the consistency between honest users

[6] Full function privacy, which in particular implies black box function privacy, *is* possible in the *secret key* setting for functional encryption [BS15].

[7] To prevent the user himself from learning f, we can imagine sk_f is generated using a multiparty computation protocol between the sever and user.

seems most important. We formalize the above observations, showing that even the weakest versions of our consistency notion are impossible for black box public tracing schemes. Our strongest notion of consistency (where the adversary gets the master secret key) is even incompatible with black box *secret* tracing schemes.

Theorem 3 (Informal)**.** *Black box publicly traceable schemes cannot be consistent.*

Challenges. We first observe that our private traitor tracing scheme is *not* consistent, a consequence of the first step of our tracing algorithm being black box. First, known UOFs for circuits are non-evasive, with the accepting inputs depending on the function. With this fact, the black box tracing step can easily be used to find differing inputs. An even more basic reason is that tracing recovers the tags τ, and keys with different tags have different functionalities. For privacy, these are not concerning since the tags and non-evasive parts of the UOFs are independent of the identifying information that must be kept secret. For consistency, however, these issues mean it is easy to find differing inputs. Even if one can find evasive UOFs for circuits, the tag problem will persist, as secret keys must have distinct tags for collusion-resistance.

Our Construction. We are unable to fully solve the consistency problem. However, in Section 6 we achieve a solution which remains traceable with public tracing for *constant-sized* collusions, and achieves consistency (and privacy) for arbitrary collusions.

At a very high level, our idea is to restructure $f_{g,\tau}$ to require a special key σ in order to activate the functionality g. By keeping σ secret, we can guarantee that differing inputs cannot be found. However, keeping σ secret means tracing is no longer possible. To overcome this issue, we encrypt σ using a fully homomorphic encryption (FHE) scheme. We can then run the tracing algorithm homomorphically on the encryption of σ, arriving at an encryption of the users' sensitive information. Of course, we now need a way to decrypt to recover the information in the clear, without using the FHE decryption key (recall that we want *public* tracing). We show that, by providing a certain compute-and-compare obfuscation in the public key [WZ17, GKW17], we can allow for decrypting in exactly the instance where tracing succeeds.

Unfortunately, the above is not consistent if the adversary has even a single key. This is because any user with a secret key can run the tracing algorithm on their key. It is not difficult to show that doing so will actually allow the user to decrypt any FHE ciphertext—including recovering σ—bring us back to square one. The natural solution is to have different σ and different FHE instances for each user, so that a user can recover their own σ but no one else's. But if the σ are isolated in different instances, there is no way to simultaneously provide the σ for different users when homomorphically running the tracing algorithm.

Our solution is to provide a unique σ and FHE instance for each *subset* of users; we set σ to simply be a signature on the (description of the) set. Of course, there are exponentially-many such subsets, so we only consider subsets of a *constant* size c to keep the number of subsets polynomial. Tracing then runs homomorphically on every subset of tags, using the compute-and-compare obfuscations to check if tracing succeeded, and if so recovering the sensitive information of the users for that subset. We prove that, as long as at most c users collude, at least one of the subsets will succeed during honest tracing.

Now, an adversary controlling a set S of secret keys will be able to run tracing on any subset of S, and would be able to find the σ for that subset. Using such σ would allow the adversary to find inputs that differ amongst the keys they control. However, we show that the σ for any set *not*

entirely contained in S remains hidden. This is sufficient to show that the adversary cannot find differing inputs for the honest users.

By instantiating our scheme with known FHE and compute-and-compare obfuscations, we obtain the following:

Theorem 4 (Informal)**.** *Assuming circularly secure learning with errors, or both sub-exponentially secure indistinguishability obfuscation and lossy encryption, for any constant c, there exists a traitor tracing scheme with public tracing, embedded identities, and consistency, tolerating c collusions.*

3 Traitor Tracing Definitions

Here, we define traitor tracing, including our new notions of privacy and consistency. Our actual constructions will be given in Sections 4, 5, and 6.

3.1 Basic Tracing Definition

We first recall the definition of (plain) traitor tracing appearing in recent works [NWZ16, GKRW18, GKW18, Zha20]. A traitor tracing scheme is a tuple $\Pi_{\mathsf{TT}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Derive}, \mathsf{Dec}, \mathsf{Trace})$ of PPT algorithms:

$$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N) \quad [8] \qquad \mathsf{sk} \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id}) \qquad m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$$
$$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \qquad A \leftarrow \mathsf{Trace}(\mathsf{pk}, \mathsf{D}, m_0, m_1, 1^N, 1^{1/\epsilon}) \ .$$

Above, λ is the security parameter, N an upper bound on the number of users, pk the public key, msk the master secret key, id a user identity, sk a user-specific secret key, m a message, and c a ciphertext, D the code of a decoder program, m_0, m_1 two challenge messages, and $\epsilon \in (0, 1/2]$ a "goodness" parameter. We require that there exists a negligible function negl such that for all $\lambda > 0, N > 0, \mathsf{id}, m$:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, c) = m : \begin{smallmatrix}(\mathsf{pk},\mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N) \\ \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id}) \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\end{smallmatrix}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

For security, consider the following experiment on adversary $\mathcal{A}$ and parameter $\epsilon = \epsilon(\lambda)$:

- $\mathcal{A}$ gets input 1^λ, and produces a number N.
- Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N)$. Send pk to $\mathcal{A}$.
- $\mathcal{A}$ then makes at most N "identity" queries on identities id. For each query, respond with $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id})$. Let T be the set of id queried.
- $\mathcal{A}$ outputs D and m_0, m_1. Run $A \leftarrow \mathsf{Trace}(\mathsf{pk}, \mathsf{D}, m_0, m_1, 1^{|T|}, 1^{1/\epsilon})$.

We define the following events. $\mathsf{BadTr}_\epsilon(\mathcal{A}, \lambda)$ means an honest user is accused: that is, $A \not\subseteq T$. $\mathsf{GoodDec}_\epsilon(\mathcal{A}, \lambda)$ means the decoder succeeds in distinguishing encryptions of m_0 and m_1: $\Pr[\mathsf{D}(c) = b : b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$. In this case, we say D is "good." $\mathsf{GoodTr}_\epsilon(\mathcal{A}, \lambda)$ means *someone* is accused: $|A| > 0$.

[8] Note that N may be allowed to be super-polynomial.

Definition 1. A traitor tracing scheme Π_{TT} is *traceable* if, for all PPT $\mathcal{A}$ and inverse-poly ϵ, there exists a negligible function negl such that $\Pr[\mathsf{BadTr}_\epsilon(\mathcal{A},\lambda)] \leq \mathsf{negl}(\lambda)$ and $\Pr[\mathsf{GoodTr}_\epsilon(\mathcal{A},\lambda)] \geq \Pr[\mathsf{GoodDec}_\epsilon(\mathcal{A},\lambda)] - \mathsf{negl}(\lambda)$.

We will occasionally distinguish between traitor tracing schemes where Trace has full access to the code of D versus schemes where Trace only makes queries to the decoder. We will say that a scheme where Trace has full access is *white box traceable*, and a scheme where Trace only makes queries is *black box traceable*; for the latter we write $\mathsf{Trace}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^N, 1^{1/\epsilon})$. We note that most prior work on traitor tracing explicitly defines tracing to be black box.

We will also consider traitor tracing with bounded collusions, where N is bounded to some value c, which may be a function of λ. In this case, we say the scheme is *c-bounded collusion traceable* (or c-traceable, for short).

Remark 5. The bulk of the tracing literature sets the identity space to be $[N]$. Starting with Nishimaki et al. [NWZ16], some recent works have considered the case where the identity space is exponentially large, say n-bit strings. These works often use terminology such as "embedded identities" [GKW19] to disambiguate from the usual setting. In this work, we will largely ignore such distinctions.

3.2 Private Traitor Tracing

We now give our new definition of privacy in traitor tracing. Let $\mathcal{A}$ be an adversary, and consider the following experiment:

- $\mathcal{A}$ gets input 1^λ, and produces a number N.
- Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N)$, and send pk to $\mathcal{A}$. $\mathcal{A}$ can now make two kinds of queries, in any order:
  - At most N "identity" queries on identities id; respond with $\mathsf{sk} \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id})$.
  - A single "challenge" query on two identities $\mathsf{id}_0^*, \mathsf{id}_1^*$. Choose a random bit $b \in \{0,1\}$ and compute $\mathsf{sk}^* \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id}_b^*)$; There is no reply.
- After the challenge query is made, $\mathcal{A}$ can additionally make arbitrary "ciphertext" queries on ciphertexts c. Respond with $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}_b^*}, c)$.
- Finally, $\mathcal{A}$ outputs a guess b' for b.

Definition 2. A traitor tracing scheme Π is *indistinguishably private* ($\mathsf{IND\text{-}P}$) if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that $\Pr[b' = b] \leq 1/2 + \mathsf{negl}(\lambda)$.

Note that our definition allows for id_b^* to also be asked during identity queries. We can also consider some variations on the above notion:

- We can limit N to be at most some value c (which may depend on λ). We say such a scheme is c-bounded collusion indistinguishably private (c-$\mathsf{IND\text{-}P}$).
- Alternatively, we can imagine giving $\mathcal{A}$ the master secret key msk in the clear at the beginning of the experiment. In this case, we note that identity queries are redundant, as the adversary can now run Derive for himself. This setting captures the case where the master secret key

may unintentionally be leaked, or alternatively where the key distributor is initially honest but later becomes corrupted. In this case, we say the scheme is *leaked master indistinguishably private* (LM-IND-P).

Simulation-based notions. We can also define *simulation*-based notions of privacy for traitor tracing, which require that the responses to ciphertext queries can be simulated without knowing the identity. More precisely, we consider two experiments, called the "Real world" and the "Ideal world". The "Real world" is identical to the experiment above, except that the challenge query consists of a single identity id^* and $\mathsf{sk}^* \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id}^*)$. In the "Ideal world", the experiment is the same, except that sk^* is never computed, and ciphertext queries on ciphertext c are answered by a simulator $S(\mathsf{msk}, c, r)$ that does not know id^*. Here, r is some randomness that is chosen at the beginning of the experiment, and used for every ciphertext query. Let W_R, W_I be the probabilities $\mathcal{A}$ outputs 1 in the Real/Ideal world, respectively.

Definition 3. A traitor tracing scheme Π is *simulation private* (SIM-P) if there exists a simulator S such that, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that $|\Pr[W_R = 1] - \Pr[W_I = 1]| \leq \mathsf{negl}(\lambda)$.

We can define c-SIM-P and LM-SIM-P analogously. Note that we always give S the master secret key msk; this is in some sense necessary, since S somehow must be able to decrypt ciphertexts in order to simulate.

It is easy to show that the indistinguishability and corresponding simulation notions are equivalent, by having S simply compute sk for an arbitrary id, and answer all decryption queries with sk. Thus we have that:

Lemma 1. *A traitor tracing scheme Π is* IND-P *(respectively c-IND-P/LM-IND-P) if and only if it is* SIM-P *(resp. c-SIM-P/LM-SIM-P).*

Impossibility of Black Box Traceable Private Traitor Tracing. Here, we show that black box traceable private traitor tracing is impossible:

Theorem 1. *If Π is black box 1-traceable*[9]*, then it is not even* 0-IND-P.

Proof. Let $\mathcal{A}$ be the following adversary for privacy: on input $\mathsf{pk}, \mathsf{msk}$, it outputs two random (distinct) identities $\mathsf{id}_0^*, \mathsf{id}_1^*$. It also chooses two random distinct messages m_0, m_1. Let $\epsilon = 1/2$. $\mathcal{A}$ runs $A \leftarrow \mathsf{Trace}^{\mathsf{Dec}}(\mathsf{pk}, m_0, m_1, 1^1, 1^{1/\epsilon})$, where Dec uses $\mathcal{A}$'s decryption oracle to decrypt ciphertexts c to get m, and then outputs 1 if and only if $m = m_1$. If A contains exactly one of id_b^*, output b; otherwise output a random bit. By the correctness of Π, with probability $1 - \mathsf{negl}$, D is a "good" decoder and so $\mathsf{GoodDec}_\epsilon(\mathcal{A}, \lambda)$ happens. As D only depends on the single secret key $\mathsf{sk}_{\mathsf{id}_b^*}$, 1-traceability means $A = \{\mathsf{id}_b^*\}$ with probability $1 - \mathsf{negl}$. As such, $\mathcal{A}$ will output b with probability $1 - \mathsf{negl}$, breaking 0-IND-P. □

3.3 Consistent Traitor Tracing

In this section, we give our new definition of consistency for traitor tracing. Let $\mathcal{A}$ be an adversary, and consider the following experiment:

[9] Recall that c-traceable means the adversary gets $\leq c$ secret keys.

- $\mathcal{A}$ gets input 1^λ, and produces a number N.
- Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N)$, and send pk to $\mathcal{A}$. $\mathcal{A}$ can now make two kinds of queries, in any order:
  - At most N "identity" queries on identities id; respond with $\mathsf{sk} \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id})$. Let T be the set of id queried.
  - A single "challenge" query on two identities $\mathsf{id}_0^*, \mathsf{id}_1^*$. Compute $\mathsf{sk}_b^* \leftarrow \mathsf{Derive}(\mathsf{msk}, \mathsf{id}_b^*)$ for $b = 0, 1$; there is no reply.

  Throughout the experiment, we require $\mathsf{id}_0^*, \mathsf{id}_1^* \notin T$, or else we immediately abort and set the output of the experiment to Lose.
- After the challenge query is made, $\mathcal{A}$ can additionally make arbitrary "ciphertext" queries on ciphertexts c. For such queries, compute $m_b = \mathsf{Dec}(\mathsf{sk}_{\mathsf{id}_b^*}, c)$ for $b = 0, 1$. If $m_0 = m_1$, respond with m_0. Otherwise, immediately abort and set the output of the experiment to Win
- At the end of the experiment, if no abort happened, output Lose.

Definition 4. A traitor tracing scheme Π is *weakly consistent* ($\mathsf{W\text{-}CONSIS}$) if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that $\Pr[\mathsf{Win}] \leq \mathsf{negl}(\lambda)$.

Note that we can consider numerous variants of consistency:

- We can bound N by c, in which case Π is *c-bounded collusion weakly consistent* (c-$\mathsf{W\text{-}CONSIS}$).
- We can give $\mathcal{A}$ the master secret key msk, in which case we say that Π is *leaked master weakly consistent* ($\mathsf{LM\text{-}W\text{-}CONSIS}$). Note that in this case, identity queries are redundant.
- Instead of having separate challenge and ciphertext queries, we can simply ask the adversary to produce a ciphertext that results in different decryption outcomes *among the secret keys controlled by the adversary.* In other words, the adversary cannot find a differing input amongst his secret keys, even though he has them in the clear. In this case, we say that Π is *strongly consistent* ($\mathsf{S\text{-}CONSIS}$). Bounded collusion strong consistency and leaked master strong consistency are defined similarly.

Relation to Privacy. Here, we discuss the relationship between privacy and consistency. We observe that consistency actually implies privacy: if the adversary cannot find a ciphertext on which two secret keys decrypt differently, then anything it can learn by querying the secret key must be independent of the identifying information. This is formalized by the following:

Theorem 5. *If* Π *is* $\mathsf{W\text{-}CONSIS}$ *(respectively* $\mathsf{LM\text{-}W\text{-}CONSIS}$ *or* c-$\mathsf{W\text{-}CONSIS}$*), then it is also* $\mathsf{IND\text{-}P}$ *(resp.* $\mathsf{LM\text{-}IND\text{-}P}$*,* c-$\mathsf{IND\text{-}P}$*).*

Proof. We prove the case $\mathsf{W\text{-}CONSIS} \Rightarrow \mathsf{IND\text{-}P}$, the other cases being proved similarly. Let $\mathcal{A}$ be an adversary for $\mathsf{IND\text{-}P}$; we use $\mathcal{A}$ to construct an adversary $\mathcal{A}'$ for $\mathsf{W\text{-}CONSIS}$. $\mathcal{A}'$ runs $\mathcal{A}$, answering all queries by forwarding all queries to its own challenger. Notice that $\mathcal{A}'$ perfectly simulates the view of $\mathcal{A}$, up until $\mathcal{A}$ makes a ciphertext query where the secret keys $\mathsf{sk}_0^*, \mathsf{sk}_1^*$ result in different outcomes. But in this case, $\mathcal{A}'$ will forward the query and win.

Suppose that $\mathcal{A}'$ does not win. Conditioned on this case, $\mathcal{A}$ learns nothing about b since its queries would be answered identically with both $\mathsf{sk}_0^*, \mathsf{sk}_1^*$. As such, the probability $\mathcal{A}$ wins is exactly $1/2$. Overall, if $\mathcal{A}'$ wins with probability ϵ, $\mathcal{A}$ wins with probability at most $1/2+\epsilon$. By the assumed W-CONSIS security, ϵ must be negligible. We conclude that IND-P holds. □

As an immediate corollary of Theorem 5, we have:

Corollary 1. *If Π is black box 1-traceable, then it is not even 0-W-CONSIS.*

4 Functional Encryption and Black Box Privacy

In this section, we discuss functional encryption and introduce a new notion of privacy called black box function privacy. Black box function private functional encryption will one of the main building blocks for our private tracing scheme, and may have applicaitons beyond the scope of this work.

A functional encryption scheme is tuple $\Pi_{\mathsf{FE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Derive}, \mathsf{Dec})$ of PPT algorithms:

$$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N) \qquad c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$$
$$\mathsf{sk}_f \leftarrow \mathsf{Derive}(\mathsf{msk}, f) \qquad o \leftarrow \mathsf{Dec}(\mathsf{sk}_f, c) \ .$$

Above, λ is the security parameter, N an upper bound on the number of users, pk the public key, msk the master secret key, f a function, sk_f a function-specific secret key, m a message, c a ciphertext, and o an output. We require that Dec recovers $f(m)$: there exists a negligible negl such that for all $\lambda > 0, N, f, m$:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, f, c) = f(m) : \begin{smallmatrix}(\mathsf{pk},\mathsf{msk})\leftarrow\mathsf{Gen}(1^\lambda,N)\\ \mathsf{sk}_f\leftarrow\mathsf{Derive}(\mathsf{msk},f)\\ c\leftarrow\mathsf{Enc}(\mathsf{pk},m)\end{smallmatrix}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

If the probability above is identically 1, we say the scheme is *perfectly* correct.

Ciphertext Indistinguishability. We now recall the "usual" definition of security for functional encryption [BSW10, O'N10], which we will call *ciphertext indistinguishability*. Consider the following experiment on an adversary $\mathcal{A}$:

- $\mathcal{A}$ gets input 1^λ, and produces a number N.
- Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N)$ and send pk to $\mathcal{A}$. $\mathcal{A}$ can now make two kinds of queries, in any order:
  - Up to N "function" queries on functions f; Return $\mathsf{sk}_f \leftarrow \mathsf{Derive}(\mathsf{msk}, f)$.
  - A single "challenge" query on a pair of messages m_0, m_1. Choose a random bit $b \in \{0,1\}$ and reply with $\mathsf{Enc}(\mathsf{pk}, m_b)$.

  The only restriction on m_0, m_1 and the various f is that $f(m_0) = f(m_1)$.
- Finally, $\mathcal{A}$ outputs a guess b' for b.

Definition 5. Π_{FE} is *adaptively ciphertext indistinguishable* (IND-C) if for all $\mathcal{A}$, there exists a negligible negl such that $\Pr[b' = b] \leq 1/2 + \mathsf{negl}(\lambda)$.

We also consider a c-bounded collusion (c-IND-C) version where $N \leq c = c(\lambda)$.

Known Results. Indistinguishability obfuscation (plus one-way functions) implies functional encryption with adaptive ciphertext indistinguishability [Wat14, ABSV15]. Public key encryption implies c-bounded collusion functional encryption for any polynomial c, where the parameters of the system grows polynomially in c [GVW12]. These schemes are perfectly correct.

4.1 Black Box Function Privacy

We now consider the privacy of f. Function privacy has been considered before (e.g. [BS15, AAB+13]), however it has always previously tried to keep f private even given sk_f. Note that for *public key* functional encryption, we can always construct from sk_f a circuit $C_f(x) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{Enc}(\mathsf{pk}, x))$ which computes f. Here, we consider a *stronger* notion of privacy, but in a weaker threat model. We do not care about hiding f from the user who holds sk_f; instead, we want to hide f from other *remote* users mounting a chosen ciphertext attack against the holder of sk_f. Now, an adversary can query $f(x)$ by querying sk_f on $\mathsf{Enc}(\mathsf{pk}, x)$. By Barak et al.'s impossibility result [BGI+01], such queries may reveal less than the actual code C_f, allowing for stronger privacy in the black box model.

Our Definition. Our formalization black box function privacy uses the Real/Ideal paradigm. Let f^* be a function and consider the following "Real" experiment $\mathsf{BB\text{-}FP\text{-}Exp}^{f^*}_{Real}(\mathcal{A}, \lambda)$ between an adversary $\mathcal{A}$ and a challenger:

- $\mathcal{A}$ gets input 1^λ, and produces a number N.
- Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N)$ and send pk *and* msk to $\mathcal{A}$. Run $\mathsf{sk}^* \leftarrow \mathsf{Derive}(\mathsf{msk}, f^*)$; sk^* is kept secret.
- $\mathcal{A}$ can now make an arbitrary number of "ciphertext" queries, where it produces a ciphertext c. In response, it gets $\mathsf{Dec}(\mathsf{sk}^*, c)$.
- Finally, $\mathcal{A}$ outputs a bit b, which is the output of the experiment.

Also consider the "Ideal" experiment $\mathsf{BB\text{-}FP\text{-}Exp}^{f^*}_{Ideal}(S, \lambda)$ for a "simulator" S. The Ideal experiment is identical to the Real experiment, except "ciphertext" queries are replaced with "function" queries, where S sends x and receives $f^*(x)$, and sk^* is never generated. Note that by giving $\mathcal{A}, S$ the master secret key, $\mathcal{A}$ and S can always compute function secret keys on its own.

Definition 6. A functional encryption scheme Π_{FE} is *black box function private* (BB-FP) if, for every PPT $\mathcal{A}$, there exists a PPT simulator S and a negligible negl such that, for every function f^*,

$$\left|\Pr[1 \leftarrow \mathsf{BB\text{-}FP\text{-}Exp}^{f^*}_{Real}(\mathcal{A}, \lambda)] - \Pr[1 \leftarrow \mathsf{BB\text{-}FP\text{-}Exp}^{f^*}_{Ideal}(S, \lambda)]\right| < \mathsf{negl}(\lambda).$$

4.2 Upgrading to Black Box Function Privacy

Now we upgrade any (perfectly correct) functional encryption scheme to be black box function private. The idea is simple: the simulator runs the adversary, decrypting any ciphertext query it to learn the input x, which it then sends as a function query. To decrypt, the simulator obtains the secret key for the identity function by using the master secret key. The potential problem is that the adversary may try to devise a ciphertext which decrypts inconsistently under the identity secret

key vs the secret key for the function f^*. To overcome this problem, we include a zero-knowledge proof of well-formedness, which combined with (perfect) correctness guarantees that decryption will be consistent.

Construction 1. Let $\Pi_{\mathsf{FE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Derive}, \mathsf{Dec})$ be a functional encryption scheme, and (P, V) be a NIZK proof system in the common reference string model. Then define $\Pi_{\mathsf{FE}}{}' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Derive}', \mathsf{Dec}')$ as follows:

- $\mathsf{Gen}'(1^\lambda)$: run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and sample a common reference string crs for (P, V). Output $\mathsf{pk}' = (\mathsf{pk}, \mathsf{crs})$ and $\mathsf{msk}' = \mathsf{msk}$.
- $\mathsf{Enc}'(\mathsf{pk}', m)$: Interpret $\mathsf{pk}' = (\mathsf{pk}, \mathsf{crs})$ and run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$ with uniform randomness r. Also run $\pi \leftarrow P(\mathsf{crs}, x)$ where x is the NP statement:

  $$\exists m, r : \mathsf{Enc}(\mathsf{pk}, m; r) = c$$

  Output $c' = (c, \pi)$.
- $\mathsf{Derive}'(\mathsf{msk}, f) = \mathsf{Derive}(\mathsf{msk}, f)$.
- $\mathsf{Dec}'(\mathsf{sk}, c')$: Interpret $c' = (c, \pi)$ and run $o \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$. Then verify the proof π. If π verifies, output o; otherwise output $\perp$.

The following theorem is straightforward given the above discussion:

Theorem 6. *If Π_{FE} is perfectly correct and* IND-C *(resp.* c-IND-C*) and (P, V) is a secure NIZK, then $\Pi_{\mathsf{FE}}{}'$ in Construction 1 is perfectly correct,* IND-C *(resp.* c-IND-C*) and* BB-FP*.*

5 Constructing Private Traitor Tracing

In this section, we give our private traitor tracing scheme with whitebox tracing. Our construction will consist of two pieces: we first build a *function-embedded* traitor tracing scheme, which allows for embedding functions, rather than identities. We then embed un-obfuscatable functions into the scheme. The result allows for tracing, while maintaining privacy.

5.1 Function-Embedded Traitor Tracing (FETT)

Here, we introduce and construct *function-embedded traitor tracing* (FETT). The rough idea of a FETT is that user secret keys have *functions* embedded in them, rather than just data. The tracing algorithm will take as an additional input x, and will output the embedded function f evaluated on x. While our ultimate goal is to construct a white box tracing scheme, this part of our construction will actually leverage prior techniques and will therefore be black box.

We will require a notion of black box function privacy analogous to functional encryption, where having oracle access to the decryption function of a secret key with f embedded is "no better than" having black box access to f itself. Black box function privacy implies that we *cannot* simply use identity-embedded traitor tracing (e.g. [NWZ16]) where we set the identity to be some (perhaps obfuscated) code of f. Indeed, in such a solution tracing would recover the code of f. Looking forward to our private tracing construction, black box function privacy allows us to embed an

un-obfusctatable function (UOF), and use the inability to learn the UOF under black box queries to argue privacy.

In order to formalize our notion of tracing, we actually break tracing into two steps that we call $\mathsf{FindTags}$ and Eval. The first step, $\mathsf{FindTags}$, extracts from a decoder a list of "tags" that were used in generating the user secret keys. The tracing guarantee insists that the recovered tags correspond to users controlled by the adversary. We note that these tags are independent of the function f. We then have a second step, Eval, which takes as input a tag and an input x, and computes $f(x)$, where f is the function embedded in the secret key associated with the given tag.

In the case of colluding users, these tags allow for disambiguating between the functions controlled by the adversary, both in the construction and also in the definition. We now give the full definition: a FETT is a tuple Π_{FETT} where:

$$\begin{aligned} (\mathsf{pk},\mathsf{msk}) &\leftarrow \mathsf{Gen}(1^\lambda, N) & \mathsf{sk} &\leftarrow \mathsf{Derive}(\mathsf{msk}, f, \tau) \\ c &\leftarrow \mathsf{Enc}(\mathsf{pk}, m) & A &\leftarrow \mathsf{FindTags}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^N, 1^{1/\epsilon}) \\ m &\leftarrow \mathsf{Dec}(\mathsf{sk}, c) & o &\leftarrow \mathsf{Eval}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^N, 1^{1/\epsilon}, \tau, x) \ . \end{aligned}$$

Above, $\lambda, \mathsf{pk}, \mathsf{msk}, \mathsf{sk}, m, c, \mathsf{D}, m_0, m_1, N, \epsilon$ are the same as in plain traitor tracing. τ is a tag from set Γ and x is an input. Correctness is similar to standard traitor tracing: there exists a negligible function negl such that for all $\lambda > 0, N, f, m$:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m : \begin{array}{c}(\mathsf{pk},\mathsf{msk})\leftarrow\mathsf{Gen}(1^\lambda,N)\\ \tau\leftarrow\Gamma\\ \mathsf{sk}\leftarrow\mathsf{Derive}(\mathsf{msk},f,\tau)\\ c\leftarrow\mathsf{Enc}(\mathsf{pk},m)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Consider the following experiment on adversary $\mathcal{A}$ and parameter $\epsilon = \epsilon(\lambda)$:

- $\mathcal{A}$ gets input 1^λ, and produces a number N.
- Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, N)$. Send pk to $\mathcal{A}$.
- $\mathcal{A}$ then makes an arbitrary number of queries on functions f_i. For each query, respond with $\mathsf{sk} \leftarrow \mathsf{Derive}(\mathsf{msk}, \tau_i)$, where $\tau_i \leftarrow \Gamma$ is chosen randomly. Let T be the set of τ_i generated.
- $\mathcal{A}$ produces a decoder D, two messages m_0, m_1, and an input x^*.
- Run $A \leftarrow \mathsf{FindTags}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^{|T|}, 1^{1/\epsilon})$.
- Additionally, let $y_i = f_i(x^*)$ and $y_i' \leftarrow \mathsf{Eval}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^{|T|}, 1^{1/\epsilon}, \tau_i, x^*)$ for each i such that $\tau_i \in A$.

We define the following events. $\mathsf{BadTr}_\epsilon(\mathcal{A}, \lambda), \mathsf{GoodDec}_\epsilon(\mathcal{A}, \lambda), \mathsf{GoodTr}_\epsilon(\mathcal{A}, \lambda)$ are as before: BadTr means $A \not\subseteq T$, $\mathsf{GoodDec}$ means $\Pr[\mathsf{D}(\mathsf{Enc}(\mathsf{pk}, m_b)) = b] \geq 1/2 + \epsilon(\lambda)$, and GoodTr means $|A| > 0$. Finally, $\mathsf{Incorrect}_\epsilon(\mathcal{A}, \lambda)$ means $y_i \neq y_i'$ for some i such that $\tau_i \in A$.

Definition 7. Π_{FETT} is *traceable* if, for all PPT $\mathcal{A}$ and inverse-poly ϵ, there exists a negligible negl such that $\Pr[\mathsf{BadTr}_\epsilon(\mathcal{A}, \lambda)] \leq \mathsf{negl}(\lambda)$, $\Pr[\mathsf{GoodTr}_\epsilon(\mathcal{A}, \lambda)] \geq \Pr[\mathsf{GoodDec}_\epsilon(\mathcal{A}, \lambda)] - \mathsf{negl}(\lambda)$, and $\Pr[\mathsf{Incorrect}_\epsilon(\mathcal{A}, \lambda)] \leq \mathsf{negl}(\lambda)$.

Note that the first two inequalities correspond to the standard tracing guarantee, with the τ_i playing the role of identities. The final inequality corresponds to the requirement that the function computed by Eval matches the function embedded in the secret key, at least on inputs computable by the adversary.

We also need a notion of black box function privacy for traitor tracing. The definition is syntactically *identical* to that of black box function privacy, and so we omit the formal definition. However, we note that the function f plays a different role in functional encryption vs traitor tracing: in functional encryption, the secret key for a function f recovers $f(m)$. In function-embedded traitor tracing, the secret key for a function f recovers m entirely. Yet in both cases it is possible to query f on arbitrary inputs, either through encryption or through tracing. Black box function privacy in both cases means, essentially, that you cannot do better than black box queries.

5.2 From BB Private FE to FETTs

We now show how to use functional encryption with black box function privacy to build function-embedded traitor tracing. Our construction is an adaptation of a construction from Nishimaki et al. [NWZ16]. Specifically, they give a variant of their main construction which achieves short ciphertexts, despite having secret keys with large embedded identities. In this work, we do not focus on the sizes of parameters, but the general structure of their construction will be useful for us.

5.2.1 The Oracle Jump-Finding Problem

Here we recall the oracle jump finding problem defined by Nishimaki et al. [NWZ16], and implicit in [BCP14]. Much of the following text is adapted or taken verbatim from [NWZ16]. Let $[a, b]_{\mathbb{R}}$ denote the set of real numbers from a to b (inclusive), and $[a, b]$ the set of integers from a to b (inclusive).

Definition 8 (The (n, q, δ, ϵ)-Jump Finding Problem)**.** A set $T \subseteq [1, 2^n]$ of q unknown points is chosen. Then, the an oracle $P : [0, 2^n] \to [0, 1]_{\mathbb{R}}$ is provided, such that:

- $|P(N) - P(0)| > \epsilon$. That is, over the entire domain, P varies significantly.
- For any $x, y \in [0, 2^n], x < y$ in interval $(x, y]$ that does not contain any points in T (that is, $(x, y] \cap T = \emptyset$), it must be $|P(x) - P(y)| < \delta$. That is, outside the points in T, P varies very little.

The task is to make queries to the oracle P and output *some* element in T.

A pictorial representation of the jump finding problem, taken from [NWZ16], is given in Figure 1.

As explained in [NWZ16], the (n, q, δ, ϵ)-Jump Finding Problem is impossible in general if $\epsilon < q\delta$, and can be inefficiently solved in the case $\epsilon \geq q\delta$ by querying every single point in the domain. They then show that, for sufficiently large ϵ relative to δ, that the problem can be solved efficiently. We omit the theorem here, and instead discuss a closely related problem that [NWZ16] also consider, which hides the oracle P inside a noisy oracle Q:

Definition 9. The (n, q, δ, ϵ) *noisy* jump finding problem is as follows. A set $T \subseteq [1, 2^n]$ of q unknown points and oracle $P : [0, 2^n] \to [0, 1]_{\mathbb{R}}$ are chosen as above. Now let $Q : [0, 2^n] \to \{0, 1\}$ be the probabilistic oracle which chooses and outputs a random bit that is 1 with probability $P(x)$,

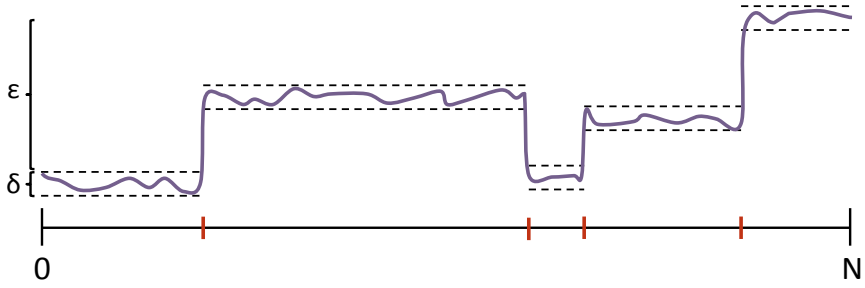

Figure 1: Example of an oracle P from [NWZ16]. Here, T contains 4 points. The purple curve represents the outputs of the oracle P on inputs in the interval $[0, 2^n]$. The red hatch marks on the number line indicate the positions of the elements in T. The horizontal dashed lines show that, between the points in T, P is never changes more than δ. At the points in T, P can make arbitrary jumps in either direction.

and 0 otherwise. A fresh sample is chosen for repeated calls to $Q(x)$, and is independent of all other samples outputted by Q. The task is to interact with the oracle Q and output *some* element in T.

Theorem 7 ([NWZ16])**.** *There is a probabilistic algorithm* $\mathsf{TraceQ}^Q(n, q, \delta, \lambda)$ *that runs in time* $t = \mathsf{poly}(n, q, 1/\delta, \lambda)$ *(and in particular makes at most* t *queries to* Q*) that will a set* A*, such that:*

- *There exists a negligible function* negl *such that, except with probability* $\mathsf{negl}(\lambda)$*,* $|P(x) - P(x-1)| \geq \delta$ *for all* $x \in A$*; in particular* $A \subseteq T$*.*
- *If* $\epsilon \geq \delta(5 + 2nq)$*, there exists a negligible function* negl *such that* A *will be non-empty except with probability* $\mathsf{negl}(\lambda)$*.*

Moreover, the algorithm works even for "cheating" P *that may not satisfy* $|P(x) - P(y)| < \delta$ *for* all $(x, y]$ *which do not intersect* T*, as long as the property holds for all pairs* x, y *that where queried by* TraceQ*.*

5.2.2 The Conversion

Construction 2. Let $\Pi_{\mathsf{FE}} = (\mathsf{Gen}_{\mathsf{FE}}, \mathsf{Enc}_{\mathsf{FE}}, \mathsf{Derive}_{\mathsf{FE}}, \mathsf{Dec}_{\mathsf{FE}})$ be a functional encryption scheme. We will assume without loss of generality that all functions considered output a single bit; we can convert any long-output function into a single-bit function by providing an additional input which selects the desired output bit. Let $\Pi_{\mathsf{FETT}} = (\mathsf{Gen}_{\mathsf{FETT}}, \mathsf{Enc}_{\mathsf{FETT}}, \mathsf{Derive}_{\mathsf{FETT}}, \mathsf{Dec}_{\mathsf{FETT}}, \mathsf{FindTags}, \mathsf{Eval})$ be the following traitor tracing scheme:

- $\mathsf{Gen}_{\mathsf{FETT}} = \mathsf{Gen}_{\mathsf{FE}}$.
- $\mathsf{Enc}_{\mathsf{FETT}}(\mathsf{pk}, m) = \mathsf{Enc}_{\mathsf{FE}}(\mathsf{pk},\ (0, \bot, m)\ )$.
- $\mathsf{Derive}_{\mathsf{FETT}}(\mathsf{pk}, g, \tau) = \mathsf{Derive}_{\mathsf{FE}}(\mathsf{pk}, f_{g,\tau})$ where $\tau \leftarrow [2^\lambda]$ and

$$f_{g,\tau}(z, x, m) = \begin{cases} m & \text{if } \tau > z \\ \bot & \text{if } (\tau < z) \vee (\tau = z \wedge x = \bot) \\ m & \text{if } \tau = z \wedge x \neq \bot \wedge g(x) = 1 \\ \bot & \text{if } \tau = z \wedge x \neq \bot \wedge g(x) = 0 \end{cases}.$$

Note that z is allowed to range from 0 to 2^λ, and x comes from the domain of g, or x can be the special symbol $\perp$.

- $\mathsf{Dec}_{\mathsf{FETT}}(\mathsf{sk}, c) = \mathsf{Dec}_{\mathsf{FE}}(\mathsf{sk}, c)$.

Before describing tracing in detail, we first give an intuition for the above construction. Under normal operation, $z = 0$ and $x = \perp$, meaning that $f_{\tau,g}$ outputs m. Therefore, the scheme is correct. We also note that black box function privacy of Π_{FETT} follows immediately from the black box function privacy of Π_{FE}.

For tracing, FindTags sets up an oracle $Q(z)$, which runs D on random FE ciphertexts encrypting index z, and outputs 1 if the decoder successfully decrypts. The oracle Q is an instance of the noisy oracle jump finding problem, with the implicit oracle P which outputs the decryption probability of the decoder on the given index. FindTags therefore runs the algorithm TraceQ from Theorem 7, to find a list of accused tags. For every such tag τ returned by TraceQ has the property that there is a significant "jump" in decryption probability between $z = \tau$ and $z = \tau - 1$.

Then Eval will set x to be the desired input and $z = \tau$, and see which side of the jump the decryption probability is closer to. Functional encryption security implies that the decoder cannot tell whether the ciphertext contains $(\tau - g(x), \perp, m)$ or (τ, x, m) (since both cases decrypt identically), and as a result the decryption probability in Eval will indicate the value of $g(x)$.

We now give the algorithms FindTags and Eval.

$\mathsf{FindTags}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^{|T|}, 1^{1/\epsilon})$. Define $P^{\mathsf{D}}(\tau) = \Pr[\mathsf{D}(\mathsf{Enc}_{\mathsf{FE}}(\tau, \perp, m_b)) = b : b \leftarrow \{0,1\}]$ as the probability that D correctly decrypts an encryption of m_b. Note that P cannot be efficiently computed, but if it *could* be computed, then $P^{\mathsf{D}}(\cdot)$ would be an instance of the oracle jump finding problem. Instead, FindTags constructs the oracle $Q^{\mathsf{D}}(\tau)$ which does the following:

- Sample $b \leftarrow \{0,1\}$ and $c \leftarrow \mathsf{Enc}_{\mathsf{FE}}(\tau, \perp, m_b)$
- Run $b' \leftarrow \mathsf{D}(c)$
- Output 1 if $b = b'$, 0 otherwise.

We see that Q^{D} is constructed from P^{D} as in the noisy jump finding problem. Next, FindTags runs and outputs $A \leftarrow \mathsf{TraceQ}^{Q^{\mathsf{D}}}(n, q, 8\delta, \lambda)$, where $n = \lambda, q = |T|$, and $8\delta = \epsilon/(5 + 2nq)$. We have the following:

Lemma 2. *Assume Π_{FE} is (IND-C), for all PPT adversaries $\mathcal{A}$ and all inverse-polynomials ϵ, there exists a negligible function* negl *such that*

- $\Pr[\mathsf{BadTr}_\epsilon(\mathcal{A}, \lambda)] \leq \mathsf{negl}(\lambda)$,
- $\Pr[\mathsf{GoodTr}_\epsilon(\mathcal{A}, \lambda)] \geq \Pr[\mathsf{GoodDec}_\epsilon(\mathcal{A}, \lambda)] - \mathsf{negl}(\lambda)$,
- *Except with probability* $\mathsf{negl}(\lambda)$, $|P(\tau) - P(\tau - 1)| \geq 8\delta$ *for all* $\tau \in A$.

Proof. For any $(x, y]$ that does not intersect T, all ciphertexts $\mathsf{Enc}_{\mathsf{FE}}(x, \perp, m), \mathsf{Enc}_{\mathsf{FE}}(y, \perp, m)$ decrypt identically under the adversary's various secret keys. As such, $P(x)$ and $P(y)$ are negligibly close, by the assumed security of the functional encryption scheme. Hence, by Theorem 7, except with

negligible probability, TraceQ outputs an A such that $|P(\tau) - P(\tau-1)| \geq 8\delta$ for all $\tau \in A$. In particular, A is a subset of T. Thus $\Pr[\mathsf{BadTr}_\epsilon(\mathcal{A}, \lambda)] \leq \mathsf{negl}(\lambda)$.

Now suppose that D is ϵ-good. Then P is an instance of the $(\lambda, q, \delta, \epsilon)$ jump finding problem (and hence Q is an instance of the corresponding noisy problem), for any inverse-polynomial $\delta' \leq \epsilon/(5+2nq)$; in particular $\delta' = 8\delta$ will do. By Theorem 7, this means that, except with negligible probability, A will be non-empty. Hence $\Pr[\mathsf{GoodTr}_\epsilon(\mathcal{A}, \lambda)] \geq \Pr[\mathsf{GoodDec}_\epsilon(\mathcal{A}, \lambda)] - \mathsf{negl}(\lambda)$. □

Next, we describe the algorithm Eval.

$\mathsf{Eval}^{\mathsf{D}}(\mathsf{aux}, \tau, x)$**.** Define $R(\tau, x) = \Pr[\mathsf{D}(\mathsf{Enc}_{\mathsf{FE}}(\tau, x, m_b)) = b : b \leftarrow \{0,1\}]$. Eval will compute estimates $\hat{P}(\tau), \hat{P}(\tau-1), \hat{R}(\tau, x)$ of $P(\tau), P(\tau-1), R(\tau, x)$ by making $O(\lambda/\delta^2)$ queries to D on ciphertexts with plaintext $(\tau, \bot, m_b), (\tau-1, \bot, m_b), (\tau, x, m_b)$, respectively. The result is:

Lemma 3. *Except with negligible probability in λ, $|\hat{P}(\tau) - P(\tau)|$, $|\hat{P}(\tau-1) - P(\tau-1)|$, and $|\hat{R}(\tau, x) - R(\tau, x)|$ are $< \delta$.*

We know that $|P(\tau_i) - P(\tau_i - 1)| \geq 8\delta$. Therefore, except with negligible probability, $|\hat{P}(\tau_i) - \hat{P}(\tau_i - 1)| \geq 6\delta$.

Now, notice that the τ_i are all distinct with overwhelming probability, and that $f_{g,\tau}(\tau, x, m) = f_{g,\tau}(\tau - g(x), \bot, m)$, while $f_{g,\tau'}(\tau, x, m)$ for $\tau' \neq \tau$ is independent of g. Therefore, by ciphertext indistinguishability, encryptions of (τ_i, x, m) are indistinguishable from encryptions of $(\tau_i - g_i(x), \bot, m)$, for any x that can be produced by the adversary. As a result, except with negligible probability, $|R(\tau_i, x) - P(\tau_i - f(x))| < \mathsf{negl}(\lambda) < \delta$. Thus, $|\hat{R}(\tau_i, x) - \hat{P}(\tau_i - f(x))| < 3\delta$, which then implies $|\hat{R}(\tau_i, x) - \hat{P}(\tau_i - (\neg f(x)))| > 3\delta$.

Therefore, Eval outputs b such that $\hat{R}(\tau_i, x)$ is closer to $\hat{P}(\tau_i - b)$ than to $\hat{P}(\tau_i - (\neg b))$. By the above, we have that $\Pr[\mathsf{Incorrect}_\epsilon(\mathcal{A}, \lambda)] \leq \mathsf{negl}(\lambda)$.

Putting everything together, we have the following theorem:

Theorem 8. *If Π_{FE} is $\mathsf{IND\text{-}C}$, then Π_{FETT} is traceable. If Π_{FE} is black box function private, then so is Π_{FETT}.*

5.3 Un-Obfuscatable Functions (UOFs)

We will make use of un-obfuscatable functions (UOFs). These were originally constructed by Barak et al. [BGI+01]. Here, we define a variant which differs in a few key ways from that of Barak et al.:

- We allow for embedding messages into the program.
- We only require that the embedded message can be reconstructed from the program code, whereas Barak et al. reconstruct the entire function.
- For technical reasons, we must allow the obfuscated code to be randomized and have differing inputs from the original code. We instead require, essentially, that as long as differing inputs are hard to find, it is possible to learn the embedded message.

More formally, an un-obfuscatable function $\Pi_{\mathsf{UOF}} = (\mathsf{Sample}, \mathsf{Extract}, \mathsf{Diff})$ is a tuple of PPT algorithms:

$$f \leftarrow \mathsf{Sample}(1^\lambda, m; r) \qquad m \leftarrow \mathsf{Extract}(f') \qquad x \leftarrow \mathsf{Diff}(f', r, m, 1^{1/\epsilon}) \ .$$

Above, m is a message, r the random coins for Sample, f' a probabilistic circuit, and ϵ a parameter. We require two security properties. First is *black box unlearnability*, where we require that it is impossible to learn anything about m with just black box access to f. Concretely, consider the following experiment on an adversary $\mathcal{A}$:

- $\mathcal{A}$, on input the security parameter 1^λ, produces two messages m_0, m_1. Choose a random bit b and compute $f \leftarrow \mathsf{Sample}(1^\lambda, m_b)$.
- $\mathcal{A}$ now can now make arbitrary queries to f.
- Finally, $\mathcal{A}$ produces a guess b' for b.

We say that Π_{UOF} is *black box unlearnable* if, for any PPT adversary $\mathcal{A}$, there exists a negligible negl such that $\Pr[b' = b] < 1/2 + \mathsf{negl}(\lambda)$.

The second security requirement is *reverse engineerability*, where we require that m can be learned given any code f' computing m. For technical reasons, we need to allow f' to be randomized and also potentially have some differing inputs from f. But if m cannot be learned, then it must be possible to actually compute such differing inputs. In more detail, consider the following experiment on adversary $\mathcal{A}$ and parameter ϵ:

- $\mathcal{A}$, on input the security parameter 1^λ, produces a message m. Reply with $f \leftarrow \mathsf{Sample}(1^\lambda, m; r)$ for random coins r.
- $\mathcal{A}$ then outputs f'. Run $m' \leftarrow \mathsf{Extract}(f')$ and $x \leftarrow \mathsf{Diff}(f', r, m, 1^{1/\epsilon})$.

Let $\mathsf{Goodfunc}_\delta(\mathcal{A}, \lambda)$ be the event that $\Pr[f'(x) \neq f(x)] < \delta(\lambda)$ where the probability in Pr is over the random coins of f'. Let $\mathsf{BadExtr}_\epsilon(\mathcal{A}, \lambda)$ be the event that f' satisfies $\Pr[\mathsf{Extract}(f') \neq m] \geq \epsilon(\lambda)$, where the probability in Pr is over the random coins on $\mathsf{Extract}$. We say that Π_{UOF} is *reverse engineerable* if, for every PPT $\mathcal{A}$ and inverse polynomial ϵ, there exists a inverse polynomial δ and negligible negl such that $\Pr[\mathsf{BadExtr}_\epsilon(\mathcal{A}, \lambda) \wedge \mathsf{Goodfunc}_\delta(\mathcal{A}, \lambda)] < \mathsf{negl}(\lambda)$. Note that in Barak et al.'s notion, f' and f are required to compute identical functions, $\mathsf{Goodfunc}$ is always guaranteed to happen and there is no need to consider Diff.

Definition 10. An un-obfuscatable function Π_{UOF} is secure if it is black box unlearnable and reverse engineerable.

5.3.1 Barak et al.'s Construction

We now recall Barak et al.'s [BGI+01] un-obfuscatable functions, and show that they satisfy our definition. The construction presented below is essentially identical to Barak et al.'s, except that we add the algorithm Diff.

Construction 3. Let (E, D) be a secret key encryption scheme for single-bit messages, and F be a pseudorandom function. Let $\Pi_{\mathsf{UOF}} = (\mathsf{Sample}, \mathsf{Extract}, \mathsf{Diff})$ be as follows:

- $\mathsf{Sample}(1^\lambda, m)$: Sample $k, s, \alpha, \beta \leftarrow \{0,1\}^\lambda$. Let $c = (c_1, \ldots, c_\lambda)$ where $c_i = \mathsf{Enc}(k, \alpha_i)$. Define

f as

$$f(j,x) = \begin{cases} c & \text{if } j = 1 \\ \beta & \text{if } j = 2 \text{ and } x = \alpha \\ \perp & \text{if } j = 2 \text{ and } x \neq \alpha \\ \mathsf{Enc}(k, b_1 \odot b_2;\ F(s,x)\ ) & \text{if } j = 3, \text{ where } \left\{ \begin{smallmatrix} x=(c_1,c_2,\odot) \\ b_1=\mathsf{Dec}(k,c_1) \\ b_2=\mathsf{Dec}(k,c_2) \end{smallmatrix} \right\} \\ m & \text{if } j = 4 \text{ and } \mathsf{Dec}(k,x) = \beta \\ \perp & \text{if } j = 4 \text{ and } \mathsf{Dec}(k,x) \neq \beta \end{cases}$$

- $\mathsf{Extract}(f')$: Choose random coins z for f' and let $h(\cdot) = f'(2, \cdot\,; z)$. Run $(c_1, \cdots, c_\lambda) \leftarrow f'(1,0)$. Note that f' allows for computing homomorphically over ciphertexts. Label the input wires to h as $1, \ldots, \lambda$, and the remaining wires $i = \lambda+1, \ldots, |h|$ such that every wire has a higher label than its children. For $i = \lambda+1, \ldots, |h|$, let $c_i \leftarrow f'(3,\ (c_{i_L}, c_{i_R}, g_i)\ )$ where i_L, i_R are the two children of wire i and g_i is the gate operation. Finally, output $m \leftarrow f'(4, \{c_i\}_{i \in o})$ where o is the list of output wires.

- $\mathsf{Diff}(f', r, m, 1^{1/\epsilon})$: Let $\delta = \epsilon/(6|f'|)$. From r, m, recover $f = \mathsf{Sample}(1^\lambda, m; r)$, including k, s, α, β. Define $p(i,x) = \Pr[f'(i,x) = f(i,x)]$, where the probability is over the random coins of f'. First, by making $O(\lambda/\epsilon^2)$ queries to f', it is possible for any (i,x) to compute estimates $\tilde{p}(i,x)$ of $p(i,x)$ such that $|\tilde{p}(i,x) - p(i,x)| < \delta$, except with non-negligible probability.

  Diff does the following. First compute the estimate $\tilde{p}(2,\alpha)$. If $\tilde{p}(2,\alpha) < 1 - 2\delta$, abort and output α.

  Next, run $\mathsf{Extract}(f')$ for $T = \lambda/\epsilon$ times. Each time $\mathsf{Extract}$ queries $f'(i,x)$, compute the estimate $\tilde{p}(i,x)$. If $\tilde{p}(i,x) < 1 - 2\delta$, abort and output (i,x).

  If no abort happens after T runs of $\mathsf{Extract}$, output an arbitrary (i,x).

Theorem 9. *If F is a secure pseudorandom function, (E,D) is secure under chosen plaintexts and non-adaptive chosen ciphertext attacks, then Π_{UOF} is secure.*

Proof. The construction is essentially identical to Barak et al., and black box unlearnability follows immediately from their proof of un-obfuscatability. For reverse engineering, note that if Diff aborts and outputs (i,x), then except with negligible probability $p(i,x) < 1 - \delta$, meaning $\mathsf{Goodfunc}_\delta$ does *not* happen. Therefore, we want to bound the probability that $\mathsf{BadExtr}_\epsilon(\mathcal{A}, \lambda)$ happens, but Diff does not abort.

Suppose $\mathsf{BadExtr}_\epsilon(\mathcal{A}, \lambda)$ happens, meaning $\Pr[\mathsf{Extract}(f') \neq m] \geq \epsilon(\lambda)$. Then we have that $\Pr_h[\Pr[\mathsf{Extract}(f') \neq m] \geq \epsilon/2] \geq \epsilon/2$, where the outer Pr is the probability over the choice of h, and the inner Pr is the probability over the remaining randomness of $\mathsf{Extract}$ once h is fixed.

Now consider running $\mathsf{Diff}(f', r, m, 1^{1/\epsilon})$, which runs $\mathsf{Extract}$ as a subroutine. Consider a run of $\mathsf{Extract}$ using an h such that $\Pr[\mathsf{Extract}(f') \neq m] \geq \epsilon/2$ for that h. With overwhelming probability, there will be at least one such h when running Diff. During such a run of $\mathsf{Extract}$, suppose Diff never aborts. This means that, except with negligible probability, every query to f' results in the correct answer with probability at least $1 - 3\delta$; the probability all queries are correct is therefore at least $1 - 3|f'|\delta$. Assuming all queries are correct, $\{c_i\}_{i \in o}$ will be an encryption of $h(\alpha)$. Then if $h(\alpha) = \beta$ and all queries are correct, $\mathsf{Extract}$ will output m. But recall that $\mathsf{Extract}$ fails to output m for such h with probability greater than $\epsilon/2 = 3|f'|\delta$. Thus we must actually have that $h(\alpha) \neq \beta$

for such h, except with negligible probability. But this means that $\Pr[h(\alpha) \neq \beta] \geq \epsilon/2 - \mathsf{negl}$; in other words $p(2,\alpha) \leq 1 - \epsilon/2 + \mathsf{negl}$. But then, except with negligible probability, we have $\tilde{p}(2,\alpha) \leq 1-\epsilon/2+\delta+\mathsf{negl} \leq \delta$, which would cause Diff to abort. Thus, Diff aborts with overwhelming probability when $\mathsf{BadExtr}_\epsilon(\mathcal{A},\lambda)$ happens. □

5.4 Our Private Traitor Tracing Scheme

We now turn to our private tracing scheme.

Construction 4. Let $\Pi_{\mathsf{FETT}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Derive}_{\mathsf{FETT}}, \mathsf{Dec}, \mathsf{FindTags}, \mathsf{Eval})$ be a FETT and $\Pi_{\mathsf{UOF}} = (\mathsf{Sample}, \mathsf{Extract}, \mathsf{Diff})$ a UOF. Define the new tracing scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Derive}, \mathsf{Dec}, \mathsf{Trace})$ where

- $\mathsf{Derive}(\mathsf{msk}, \mathsf{id})$: Return $\mathsf{sk} \leftarrow \mathsf{Derive}_{\mathsf{FETT}}(\mathsf{msk},\ \mathsf{Sample}(1^\lambda, \mathsf{id})\ )$.
- $\mathsf{Trace}(\mathsf{pk}, \mathsf{D}, m_0, m_1, 1^N, 1^{1/\epsilon})$: Produce $A \leftarrow \mathsf{FindTags}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^N, 1^{1/\epsilon})$. Then, for each $\tau_i \in A$, define P_i as the (randomized) program $x \mapsto \mathsf{Eval}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^N, 1^{1/\epsilon}, \tau_i, x)$. Now run $\mathsf{id}_i \leftarrow \mathsf{Extract}(P_i)$ for each τ_i, and output $\{\mathsf{id}_i : \tau_i \in A\}$.

The correctness of Π is immediate. We now discuss security:

Theorem 10. *If Π_{FETT} is traceable and black box function private and if Π_{UOF} is secure, then Π is traceable and leaked master indistinguishably private.*

Proof. First, we consider privacy. By black box function privacy, the view of a privacy adversary can be simulated by making black box queries to the functions f. But by the black box unlearnability of Π_{UOF}, such queries reveal nothing about the identities. Thus, privacy follows.

We now consider tracing. Let $\mathcal{A}$ be a tracing adversary for Π. We construct a new adversary $\mathcal{A}'$ for Π_{FETT}. $\mathcal{A}'$ simulates $\mathcal{A}$. For each secret key query on identity id_i, $\mathcal{A}'$ runs $f_i \leftarrow \mathsf{Sample}(1^\lambda, \mathsf{id}_i; r_i)$ with fresh randomness r_i and makes a secret key query on f_i, which it forwards to $\mathcal{A}$. $\mathcal{A}'$ thus perfectly simulates the view of $\mathcal{A}$. Now, once $\mathcal{A}$ produces a decoder D, $\mathcal{A}'$ outputs D. Additionally, $\mathcal{A}'$ runs the (public) algorithm $\mathsf{FindTags}$ on D to collect a set A' of tags τ_i. Next, $\mathcal{A}'$ matches the τ_i to the r_i, id_i, and constructs the programs P_i as in Trace. It then runs $x_i \leftarrow \mathsf{Diff}(P_i, r_i, \mathsf{id}_i, 1^{1/\epsilon'})$ for an ϵ' to be specified later. It outputs D and sets x^* to be a random choice amongst the x_i.

Consider running Trace on D, and let A be the set of τ_i recovered by $\mathsf{FindTags}$ when run as a part of Trace. By the tracing security of Π_{FETT}, we know that, except with negligible probability, $\mathsf{FindTags}$ correctly outputs a subset of the tags τ_i generated during the adversary's identity queries. It remains to show that Trace correctly recovers the corresponding id_i. Let p be the probability that A is indeed a subset of the correct τ_i, but for some i^*, the recovered identity $\mathsf{id}'_{i^*} := \mathsf{Extract}(P_{i^*})$ is such that $\mathsf{id}'_{i^*} \neq \mathsf{id}_{i^*}$. We must show that p is negligible. Suppose toward contradiction p is non-negligible, and let ϵ' be a polynomial which lower bounds $(p/N)^2$ infinitely often; note that p is determined entirely by $\mathcal{A}, \epsilon$, meaning we can set ϵ' freely.

Suppose we choose i^* at random from A; then with probability at least p/N, we will have that $\mathsf{id}'_{i^*} \neq \mathsf{id}_{i^*}$. Note also that, once $\mathcal{A}$ outputs the decoder D, A and the A' generated by $\mathcal{A}'$ are two samples from identical distributions. As a consequence, with probability at least $(p/N)^2 \geq \epsilon'$, the x^* produced by $\mathcal{A}'$ is equal to x_{i^*} and $\mathsf{id}'_{i^*} \neq \mathsf{id}_{i^*}$. Moreover, the P_{i^*} constructed by $\mathcal{A}'$ is identical to the P_{i^*} constructed inside Trace. Therefore, by the reverse engineerability of Π_{UOF}, we must have that x^* is a differing input: there exists a δ such that $\Pr[P_{i^*}(x^*) \neq f(x^*)] \geq \delta$. This implies that, in the FETT experiment, $y_{i^*} := f_{i^*}(x^*)$ and $y'_{i^*} := P_{i^*}(x^*)$ differ with non-negligible probability. In other words, $\Pr[\mathsf{Incorrect}_\epsilon(\mathcal{A}', \lambda)]$ is non-negligible, contradicting the security of Π_{FETT}. □

6 Toward Consistent Traitor Tracing

Here, we give our construction of consistent traitor tracing. We first recall some additional definitions.

Fully Homomorphic Encryption. Fully homomorphic encryption (FHE) can be built from circularly-secure variants of LWE [Gen09, GSW13], or from sub-exponentially secure iO and "lossy" encryption [CLTV15]. An FHE scheme is a tuple $\Pi_{\mathsf{FHE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ where $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ form a public key encryption scheme. Additionally, $\mathsf{Eval}(c, f)$ takes as input a ciphertext and a circuit f, with the property that, for every polynomial poly, there exists a negligible function such that, for every $\lambda > 0, x$ and f such that $|f| \leq \mathsf{poly}(\lambda)$:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},\ \mathsf{Eval}(c, f)\ ) = f(x) : \begin{smallmatrix}(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{Gen}(1^\lambda)\\ c\leftarrow\mathsf{Enc}(\mathsf{pk},x)\end{smallmatrix}\right] \geq 1 - \mathsf{negl}(\lambda)\ .$$

Compute and Compare Obfuscation. Compute and compare obfuscation can be built from LWE [GKW17, WZ17], or seen as a special case of iO. It consists of a PPT algorithm $\mathsf{CCObf}(C, \beta)$ which takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\lambda$, and a value $\beta \in \{0,1\}^\lambda$. It outputs a circuit $C' : \{0,1\}^n \to \{0,1\}$ such that:

- Correctness: $C'(x) = 1$ if $C(x) = \beta$, and $C'(x) = 0$ otherwise.
- Security: There exists an algorithm Sim with the following guarantee. For any C, let β be chosen uniformly from $\{0,1\}^\lambda$. Then for any PPT $\mathcal{A}$, there exists a negligible negl such that

  $$\left|\Pr[\mathcal{A}(\mathsf{CCObf}(C, \beta)) = 1] - \Pr[\mathcal{A}(\mathsf{Sim}(1^{|C|}, 1^n, 1^\lambda)) = 1]\right| < \mathsf{negl}(\lambda)\ .$$

  In other words, if β is uniform, then C' reveals nothing about C, β.

6.1 The Construction

We now give our construction. The intuition behind our construction is to essentially have a *secret* tracing algorithm, which requires certain secrets in order to operate. In such a setting, it is not difficult to devise a consistent traitor tracing scheme. The challenge is to somehow allow the tracer to recover the result of the secret tracing, without compromising consistency.

Towards a solution, we homomorphically encrypt the secret tracing key. This allows tracing to be public, except that the result of tracing remains encrypted under the homomorphic encryption scheme. To rectify this, we can imagine obfuscating a program which takes as input an encryption of the tracing result, and returns the result in the clear. The good news is that, by carefully setting things up, we can use compute-and-compare obfuscation exactly for this task. The result is that any remote outside user will be unable to learn the tracing results (since he cannot trace homomorphically without the actual decoder code), while anyone in possession of the decoder's code can trace in the clear.

The bad news is that the construction is inherently insecure against any user of the system. We show a simple attack which, given the compute-and-compare program and a single user's key, any FHE ciphertext can be decrypted, including the encrypted tracing secret. This applies regardless of the particular fully homomorphic encryption scheme or particular obfuscation scheme used. The intuition is that the user can run the tracing algorithm on themselves. Since tracing succeeds (their key represents a good decoder, after all), this implies that they learn an accepting input to the

compute-and-compare program. Certainly this violates the security guarantee as defined above and proven in [GKW17, WZ17]. But even with a best-possible obfuscation scheme, security still fails. Basically, using the homomorphic properties of the encryption scheme, one can decrypt any ciphertext bit-by-bit: homomorphically overriding a particular bit of the plaintext, and see if tracing still succeeds. If so, then the overridden bit is correct; otherwise it is incorrect.

Our solution. Our solution is to have a separate tracing secret, FHE instance, and compute-and-compare obfuscation for each user. When we move to the case of colluding users, the problem will be that the different secrets are isolated in different FHE instances, meaning there is no way to homomorphically tracing, let alone trace in the clear, in the case that even two users collude. Our solution is instead to have a separate secret for each *subset* of users. More precisely, for each set S of users, a *signature* on S, denoted σ^S will play the role of tracing secret. Our system is set up so that, when using the secret σ^S, users outside of S will have no differing input. Users inside of S, on the other hand, will have differing inputs that are used to trace.

Given σ^S, it will be possible to trace a decoder built using secret keys from exactly this subset of users. The result of tracing is that the tracer learns some other secret β^S associated with the set S. β^S will be derived using a pseudorandom function. We therefore encrypt each σ^S under different FHE instances. We additionally provide, for each S, a compute-and-compare program which accepts encryptions (under that FHE instance) of β^S. If S has at least one honest user, tracing will fail to recover β^S, and the compute-and-compare obfuscation will reject. On the other hand, a decoder built from keys in set S will trace (homomorphically) to β^S, which will cause the compute-and-compare obfuscation to accept. Tracing therefore homomorphically traces all possible subsets S, and accuses whichever one accepts.

For consistency, the security of compute-and-compare obfuscation means that the FHE instance for set S remains secure if there is any honest user in S, and therefore σ^S remains hidden to the adversary. Thus, an attacker can only learn σ^S for S that are entirely comprised of malicious users. Since σ^S only allows for finding differing inputs between users in S, we therefore have that the adversary can only find differing inputs for keys they control.

The problem with our construction is that we need different components for each set S, and moreover that our tracing algorithm has to try every set S. In general, there will be exponentially-many sets S, meaning our scheme is not polynomially efficient. Instead, we restrict to subsets of constant size, so that the number of subsets is only a polynomial.

Construction 5. Fix a constant c. Let $\Pi_{\mathsf{FHE}} = (\mathsf{Gen}_{\mathsf{FHE}}, \mathsf{Enc}_{\mathsf{FHE}}, \mathsf{Dec}_{\mathsf{FHE}}, \mathsf{Eval}_{\mathsf{FHE}})$ be a fully homomorphic encryption scheme, $\Pi_{\mathsf{FE}} = (\mathsf{Gen}_{\mathsf{FE}}, \mathsf{Enc}_{\mathsf{FE}}, \mathsf{Derive}_{\mathsf{FE}}, \mathsf{Dec}_{\mathsf{FE}})$ a functional encryption scheme, $\Pi_{\mathsf{Sig}} = (\mathsf{Gen}_{\mathsf{Sig}}, \mathsf{Sign}, \mathsf{Ver})$ a signature scheme, F a PRF, and CCObf a compute and compare obfuscation scheme. Define the traitor tracing scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Derive}, \mathsf{Dec}, \mathsf{Trace})$ as follows:

- $\mathsf{Gen}(1^\lambda, N)$: run $(\mathsf{pk}_{\mathsf{FE}}, \mathsf{msk}_{\mathsf{FE}}) \leftarrow \mathsf{Gen}_{\mathsf{FE}}(1^\lambda, N)$. Also run $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow \mathsf{Gen}_{\mathsf{Sig}}(1^\lambda)$. For each $i \in [N]$, choose a random key $k^i \leftarrow \{0,1\}^\lambda$. Then, for every $S \subseteq [N]$ such that $|S| \leq c$, let $\sigma^S \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, S)$, run $(\mathsf{pk}^S, \mathsf{sk}^S) \leftarrow \mathsf{Gen}_{\mathsf{FHE}}(1^\lambda)$, $c^S \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}^S, \sigma^S)$, $\beta^S \leftarrow \oplus_{i \in S} \mathsf{F}(k^i, S)$ and finally $P^S \leftarrow \mathsf{CCObf}(\mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}^S, \cdot), \beta^S)$. Output $\mathsf{pk} = (\mathsf{pk}_{\mathsf{FE}}, (\mathsf{pk}^S, c^S, P^S)_S)$ and $\mathsf{msk} = (\mathsf{msk}_{\mathsf{FE}}, (k^i)_{i \in [N]}, \mathsf{ctr} = 0)$.
- $\mathsf{Enc}(\mathsf{pk}, m) = \mathsf{Enc}_{\mathsf{FE}}(\mathsf{pk}_{\mathsf{FE}},\ (\bot, \bot, \bot, \bot, m)\ )$

- $\mathsf{Derive}(\mathsf{msk}, \mathsf{id})$: Run $\mathsf{sk} \leftarrow \mathsf{Derive}_{\mathsf{FETT}}(\mathsf{msk}_{\mathsf{FE}}, h_{\mathsf{id},i})$ where $i = \mathsf{ctr}$ and

$$h_{\mathsf{id},i}(S, \sigma, j, x, m) = \begin{cases} m & \text{if } \mathsf{Ver}(\mathsf{pk}_{\mathsf{Sig}}, S, \sigma) = 0 \vee i \in S \setminus \{j\} \\ g_{\mathsf{id},i}(S, j, x, m) & \text{if } \mathsf{Ver}(\mathsf{pk}_{\mathsf{Sig}}, S, \sigma) = 1 \wedge i \notin S \setminus \{j\} \end{cases},$$

$$g_{\mathsf{id},i}(S, j, x, m) = \begin{cases} \bot & \text{if } i \neq j \vee x = \bot \\ m & \text{if } i = j \wedge f_{\mathsf{id},i}(S, x) = 1 \\ \bot & \text{if } i = j \wedge f_{\mathsf{id},i}(S, x) = 0 \end{cases}, \text{ and}$$

$$f_{\mathsf{id},i}(S, x = (b, u)) = \begin{cases} \mathsf{F}(k^i, S)_u & \text{if } b = 0 \\ \mathsf{id}_u & \text{if } b = 1 \end{cases}.$$

  Also increment ctr within msk. Here, $\mathsf{F}(k^i, S)_u$ is the uth bit of $\mathsf{F}(k^i, S)$.

Remark 6. We note that our construction requires a stateful Gen, which keeps a counter. This is to ensure that the tags used for different users are unique. An alternative, similar to what was done in [GKW19], would be to have Derive take the tag as an explicit input, and assume some external mechanism to ensure distinct tags.

Before formally giving the tracing algorithm and proving security, we discuss the intuition behind the above construction in more detail. Consider encryptions of plaintexts $(S, \sigma^S, \bot, \bot, m)$. Since $g_{\mathsf{id},i}(S, \bot, \bot, m) = \bot$, such ciphertexts can be decrypted by users with indices $i \in S$, but cannot be decrypted by $i \notin S$. Let $p(S)$ be the probability a decoder decrypts such ciphertexts.

For any good decoder, we must have $p([N])$ be large. FE security implies $p(Q)$ is close to $p([N])$, where Q is the set of indices the adversary controls. Moreover, FE security implies that $p(\emptyset)$ is close to 0. A straightforward argument implies that there must therefore exist a set $S^* \subseteq Q$ such that $p(S^*)$ is noticeably larger than $p(S^* \setminus \{i\})$ for all $i \in S^*$. Supposing we could sign arbitrary sets, we can recover S^* by estimating the various $p(S)$ values.

On the other hand, the ability to sign sets S also allows for easily finding differing inputs, which would break consistency. Instead, we can use the encryptions of signatures to homomorphically compute $p(S)$. Unfortunately, we cannot directly compare different $p(S)$, since the signatures for different S, and therefore the $p(S)$, are isolated in different FHE instances. However, given an (encrypted) signature on S, we can (homomorphically) estimate $p(S \setminus \{i\})$ for any $i \in S$ by testing the decoder on ciphertexts encrypting $(S, \sigma^S, i, \bot, m)$. Indeed, these ciphertexts can only be decrypted by users in $S \setminus \{i\}$, and functional encryption security implies that encryptions of $(S, \sigma^S, i, \bot, m)$ are indistinguishable from encryptions of $(S \setminus \{i\}, \sigma^{S \setminus \{i\}}, \bot, \bot, m)$.

For the set S^* (which at this point is still FHE encrypted), we can then run the decoder (again, homomorphically) on encryptions of $(S^*, \sigma^{S^*}, i, x, m)$. Depending on the value of $f_{\mathsf{id},i}(S^*, x)$, the decryption probability will either be roughly $p(S^*)$ or $p(S^* \setminus \{i\})$; since the definition of S^* means the two probabilities are noticeably different, this allows us to learn $f_{\mathsf{id},i}(S^*, x)$. From here, we can compute $\mathsf{F}(k^i, S^*)$ by setting $b = 0$, and hence β^{S^*}.

Up until this point, we cannot actually tell which set is S^*, since all results are computed homomorphically and therefore still hidden under FHE encryptions. We perform the above procedure for each set S, as there are only polynomial many. We then apply the program P^S to the resulting ciphertext, which will output 1 in the case $S = S^*$. This allows us to actually determine S^*.

The next step is to determine the identities for users in S^*. We show that any accepting input to P^{S^*} actually allows us to decrypt ciphertexts encrypted under pk^{S^*}; in particular we can find σ^{S^*} in the clear. This then allows us to evaluate $f_{\mathsf{id},i}(S^*, x)$ on arbitrary inputs x in the clear. Using such queries we can easily compute the various id by setting $b = 1$.

It remains to justify consistency, which follows from the fact that β^S is pseudorandom as long as S contains honest users. By applying compute and compare security, we have that σ^S remains hidden for such sets. Since the adversary cannot obtain a signature on any S containing honest users, any ciphertext he devises will be decrypted correctly by all honest users; in particular, all honest users answer identically.

6.2 Tracing

We now give the algorithm $\mathsf{Trace}(\mathsf{pk}, \mathsf{D}, m_0, m_1, 1^c, 1^{1/\epsilon})$. Define

$$\begin{aligned} p(S) &= \Pr[\mathsf{D}(\mathsf{Enc}_{\mathsf{FE}}(\mathsf{pk}_{\mathsf{FE}}, (S, \sigma^S, \perp, \perp, m_b))) = b : b \leftarrow \{0,1\}] \\ p(S,i) &= \Pr[\mathsf{D}(\mathsf{Enc}_{\mathsf{FE}}(\mathsf{pk}_{\mathsf{FE}}, (S, \sigma^S, i, \perp, m_b))) = b : b \leftarrow \{0,1\}] \\ q(S,i,x) &= \Pr[\mathsf{D}(\mathsf{Enc}_{\mathsf{FE}}(\mathsf{pk}_{\mathsf{FE}}, (S, \sigma^S, i, x, m_b))) = b : b \leftarrow \{0,1\}] \ . \end{aligned}$$

Let $\delta = \epsilon/(10c+2)$. For any S, given σ^S we can compute an estimates $\tilde{p}(S), \tilde{p}(S,i), \tilde{q}(S,i,x)$ such that $|\tilde{p}(S) - p(S)|, |\tilde{p}(S,i) - p(S,i)|, |\tilde{q}(S,i,x) - p(S,i,x)| < \delta$, except with negligible probability. Each quantity is computed by making $O(\lambda/\delta^2)$ queries to D. We define several subroutines:

- $\mathsf{ConfirmTags}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, S, \sigma)$: This algorithm plays an analogous role as $\mathsf{FindTags}$ from Section 5, except that instead of discovering a set of accused users, it simply confirms whether the input set S should be accused. In this sense, $\mathsf{ConfirmTags}$ works in the black box confirmation model of [BF99]. The algorithm is also somewhat different that that of Section 5, owing to the different tracing structure in this construction.

  Compute estimate $\tilde{p}(S)$, and for each $j \in S$, compute estimates $\tilde{p}(S,j)$. If there exists a $j \in S$ such that $|\tilde{p}(S,j) - \tilde{p}(S)| < 4\delta$, abort and output $\perp$. Otherwise, output $\mathsf{aux} = (\tilde{p}(S), (\tilde{p}(S,j))_{j \in S})$.

- $\mathsf{Eval}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, \mathsf{aux}, S, \sigma, j, x)$: This algorithm is analogous to Eval from Section 5. Compute estimate $\tilde{q}(S,j,x)$, and output 1 such that $\tilde{q}(S,j,x)$ is closer to $\tilde{p}(S)$ than it is to $\tilde{p}(S,j)$; otherwise output 0.

- $\mathsf{Dec}^*(\mathsf{pk}^S, c^S, P^S, C, d)$: here, C is a circuit, with the property that $C(\sigma^S) = \beta^S$, meaning $P^S(\mathsf{Eval}_{\mathsf{FHE}}(c^S, C)) = 1$; d is a ciphertext encrypting a bit b. Dec^* will output b. Dec^* works as follows. It homomorphically computes d', an encryption of $b \cdot \sigma^S$, from d, c^S. Then it will run and output $P^S(\mathsf{Eval}_{\mathsf{FHE}}(d', C))$.

With these subroutines in hand, Trace works as follows. Let $C^S(\sigma)$ be the function which runs $\mathsf{ConfirmTags}$ to recover aux or $\perp$. If aux is recovered, then for each $i \in S$, it runs the algorithm $\mathsf{Eval}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, \mathsf{aux}, S, \sigma, i, x)$ on the various $x = (0, u)$ to compute strings $\beta^{S,i} = \mathsf{F}(k^i, S)$. Finally, it outputs $\oplus_{i \in S} \beta^{S,i}$. The circuit C^S is ostensibly randomized, but we will hard-code the randomness to get a deterministic circuit.

For each set S, we will say that tracing succeeds if $P^S(\mathsf{Eval}(c^S, C^S)) = 1$, which is equivalent to requiring $C^S(\sigma^S) = \beta^S$. For each S, Trace runs $\mathsf{Dec}^*(\mathsf{pk}^S, c^S, P^S, C^S, d = c^S)$. Let S be some set

such that Dec^* outputs a signature σ. Then Trace runs $\mathsf{ConfirmTags}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, S, \sigma)$ to recover aux, and runs $\mathsf{Eval}^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, \mathsf{aux}, S, \sigma, i, x)$ on various $x = (1, u)$ to compute the bits of id^i for $i \in S$.

Remark 7. Note that our tracing algorithm is homomorphically running the decoder algorithm. In general, the decoder will have no a priori polynomial bound. As such, we need the full power of FHE, and cannot rely on "leveled" FHE, which only supports an a priori bounded computation.

6.3 Security

Theorem 11. *If $\Pi_{\mathsf{FHE}}, \Pi_{\mathsf{Sig}}, \mathsf{F}, \mathsf{CCObf}$ are secure, and Π_{FE} is both ciphertext indistinguishable and black box function private, then Π in Construction 5 is c-traceable and (unbounded) weakly consistent.*

Proof. We first prove weak consistency. Let $\mathcal{A}$ be an adversary for consistency. By the black box function privacy of Π_{FE}, there exists a simulator Sim that only makes queries to the functions $h_{\mathsf{id},i}$ of the various honest users, and can still find a differing input with non-negligible probability. In particular, it must with non-negligible probability find a query (S^*, σ, z, x, m) to some $h_{\mathsf{id},i}$ such that σ is a valid signature on S^* and $i \in S^*$. Let q be the index of the first query where this happens. For all prior queries, $h_{\mathsf{id},i}$ outputs m. Therefore, all prior queries can be simulated without knowing a signature on any S that contains honest users, and also without knowing k^i for any honest user i.

For every honest user i, we can therefore replace each evaluation of $\beta^{S,i}$ with random. This change will be undetectable before query q, by the PRF security of F. But this means, by compute and compare security, that $P^{S,i}$ can be simulated without knowing $\mathsf{sk}^{S,i}$, which again will be undetectable before query q. We finally rely on the security of $\mathsf{pk}^{S,i}$ to conclude that the entire view of the adversary up until query q can be simulated just knowing σ^S, where S ranges over all subsets containing only adversarial users.

The result is that the view of the adversary up until query q can be simulated by making signing queries on S containing only adversarial users, but then query q produces a signature on an S^* containing at least one honest user, which must therefore be different than any of the queries S. Thus, such an algorithm can forge signatures, a contradiction to the security of $\mathsf{pk}_{\mathsf{Sig}}$.

We now prove c-traceability. Consider an attacker $\mathcal{A}$ which makes up to c queries. Let Q be the set of $i \in [N]$ corresponding to the adversary's queries. Let D be the output of $\mathcal{A}$, and suppose $\mathsf{GoodDec}_\epsilon$ happens. We note that for any honest user $i \notin Q$, by functional encryption security $p(S)$ and $p(S \setminus \{i\})$ will be negligibly close, except with negligibly-small probability. As such, honest users will never be accused. We now prove that some user will be accused.

Claim 1. *Except with negligible probability, $p(Q) > 1/2 + \epsilon - \delta$ and $p(\emptyset) < 1/2 + \delta$*

Proof. Under all the adversary's keys, $(\perp, \perp, \perp, \perp, m)$ and $(Q, \sigma^Q, \perp, \perp, m)$ decrypt correctly, so encryptions of these values are indistinguishable. $p(Q) > 1/2 + \epsilon - \delta$ follows by the goodness of D. On the other hand, $(\emptyset, \sigma^\emptyset, \perp, \perp, m)$ will always fail to decrypt, so $p(\emptyset) < 1/2 + \delta$ except with negligible probability. □

Claim 2. *Except with negligible probability, there exists an $S^* \subseteq Q$ such that, for all $i \in S^*$, $p(S^* \setminus \{i\}) \leq p(S^*) - 8\delta$.*

Proof. Assume $p(Q) > 1/2 + \epsilon - \delta$ and $p(\emptyset) < 1/2 + \delta$. Suppose toward contradiction that, for each set $S \subseteq Q$, there exists an i_S such that $p(S \setminus \{i_S\}) > p(S) - 8\delta$. Then setting $S_0 = Q$ and $S_j = S_{j-1} \setminus \{i_{S_{j-1}}\}$, we get that $p(S_j) > p(S_{j-1}) - 8\delta$ and $S_{|Q|} = \emptyset$. But this means that $p(\emptyset) > p(Q) - 8\delta|Q|$, a contradiction. □

Claim 3. *Except with negligible probability, $|p(S^* \setminus \{j\}) - p(S^*, j)| < \delta$ for any $i \in S^*$.*

Proof. For any $i \in S^*$ and any secret key under the adversary's control, $(S^* \setminus i, \sigma^{S^* \setminus i}, \bot, \bot, m)$ and $(S^*, \sigma^{S^*}, i, \bot, m)$ decrypt identically. Therefore, their encryptions are indistinguishable. □

By the above claims, $p(S^*, i) < p(S^*) - 8\delta$ for each $i \in S^*$, except with negligible probability. But then $\tilde{p}(S^*, i) < \tilde{p}(S^*) - 6\delta$ except with negligible probability. When we homomorphically run $\mathsf{FindTags}_0^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, S^*, \sigma^*)$, no abort will happen and the result will be (an encryption of) aux.

Claim 4. *For any $i \in S^*$ and x, if $f_{\mathsf{id},i}(S^*, x) = 0$ then $|q(S^*, i, x) - p(S^*, i))| < \delta$ except with negligible probability. If $f_{\mathsf{id},i}(S^*, x) = 1$, then $|q(S^*, i, x) - p(S^*))| < \delta$.*

Proof. If $f_{\mathsf{id},i}(S^*, x) = 0$, then the secret key for user i rejects encryptions of $(S^*, \sigma^{S^*}, i, x, m)$, while all other users in S^* decrypt and users outside S^* reject. This is the same functionality as $(S^*, \sigma^{S^*}, i, \bot, m)$. On the other hand, if $f_{\mathsf{id},i}(S^*, x) = 1$, then the secret key for user i correctly decrypts $(S^*, \sigma^{S^*}, i, x, m)$, corresponding to the same functionality as $(S^*, \sigma^{S^*}, \bot, \bot, m)$. The claim follows by functional encryption security. □

Claim 5. *For any $i \in S^*$ and any input x, $\mathsf{Eval}_0^{\mathsf{D}}(\mathsf{pk}, m_0, m_1, 1^c, 1^{1/\epsilon}, \mathsf{aux}, S^*, \sigma^{S^*}, i, x)$ outputs $f_{\mathsf{id},i}(S^*, x)$, except with negligible probability.*

Proof. If $f_{\mathsf{id},i}(S^*, x) = 0$, then $|\tilde{q}(S^*, i, x) - \tilde{p}(S^*, i)| < 3\delta$. But since $|\tilde{p}(S^*) - \tilde{p}(S^*, i)| > 6\delta$, we must have $|\tilde{q}(S^*, i, x) - \tilde{p}(S^*)| > 3\delta$. As such, $\tilde{q}(S^*, i, x)$ is closer to $\tilde{p}(S^*, i)$ than $\tilde{p}(S^*)$, and Eval_0 therefore outputs 0 on input x. Analogously, Eval_0 outputs 1 on inputs x such that $f_{\mathsf{id},i}(S^*, x) = 1$. □

Therefore, the circuit $C^{S^*}(\sigma^{S^*})$ will correctly evaluate $\beta^{S^*,i} = \mathsf{F}(k^i, S^*)$, and therefore correctly output β^{S^*} with overwhelming probability. □

References

[AAB+13] Shashank Agrawal, Shweta Agrawal, Saikrishna Badrinarayanan, Abishek Kumarasubramanian, Manoj Prabhakaran, and Amit Sahai. Functional encryption and property preserving encryption: New definitions and positive results. Cryptology ePrint Archive, Report 2013/744, 2013. http://eprint.iacr.org/2013/744.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[ADVW13] Shweta Agrawal, Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. On continual leakage of discrete log representations. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 401–420. Springer, Heidelberg, December 2013.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014.

[BF99] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 338–353. Springer, Heidelberg, August 1999.

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

[BN08] Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 501–510. ACM Press, October 2008.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 241–250. ACM Press, June 2013.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 306–324. Springer, Heidelberg, March 2015.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 573–592. Springer, Heidelberg, May / June 2006.

[BSW10] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. Cryptology ePrint Archive, Report 2010/543, 2010. http://eprint.iacr.org/2010/543.

[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, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 257–270. Springer, Heidelberg, August 1994.

[CGZ20] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 828–858. Springer, Heidelberg, May 2020.

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1115–1127. ACM Press, June 2016.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[DF03] Yevgeniy Dodis and Nelly Fazio. Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 100–115. Springer, Heidelberg, January 2003.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GKRW18] Rishab Goyal, Venkata Koppula, Andrew Russell, and Brent Waters. Risky traitor tracing and new differential privacy negative results. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 467–497. Springer, Heidelberg, August 2018.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, October 2017.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 660–670. ACM Press, June 2018.

[GKW19] Rishab Goyal, Venkata Koppula, and Brent Waters. New approaches to traitor tracing with embedded identities. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 149–179. Springer, Heidelberg, December 2019.

[GNPT13] Philippe Guillot, Abdelkrim Nimour, Duong Hieu Phan, and Viet Cuong Trinh. Optimal public key traitor tracing scheme in non-black box model. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13*, volume 7918 of *LNCS*, pages 140–155. Springer, Heidelberg, June 2013.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[JKL09] Pascal Junod, Alexandre Karlov, and Arjen K. Lenstra. Improving the Boneh-Franklin traitor tracing scheme. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 88–104. Springer, Heidelberg, March 2009.

[KD98] Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 145–157. Springer, Heidelberg, May / June 1998.

[KY03] Aggelos Kiayias and Moti Yung. Breaking and repairing asymmetric public-key traitor tracing. In Joan Feigenbaum, editor, *Digital Rights Management*, pages 32–50, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[NDC+15] J. Ning, X. Dong, Z. Cao, L. Wei, and X. Lin. White-box traceable ciphertext-policy attribute-based encryption supporting flexible attributes. *IEEE Transactions on Information Forensics and Security*, 10(6):1274–1288, 2015.

[NP01] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In Yair Frankel, editor, *FC 2000*, volume 1962 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2001.

[NWZ16] Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 388–419. Springer, Heidelberg, May 2016.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556.

[Pfi96] Birgit Pfitzmann. Trials of traced traitors. In Ross Anderson, editor, *Information Hiding*, pages 49–64, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[TS06] Dongvu Tonien and Reihaneh Safavi-Naini. An efficient single-key pirates tracing scheme using cover-free families. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 82–97. Springer, Heidelberg, June 2006.

[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.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th FOCS*, pages 600–611. IEEE Computer Society Press, October 2017.

[Zha20] Mark Zhandry. New techniques for traitor tracing: Size $N^{1/3}$ and more from pairings. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 652–682. Springer, Heidelberg, August 2020.