Traceable Secret Sharing Revisited

Vipul Goyal
NTT Research

Abhishek Jain
NTT Research & JHU

Aditi Partap
Stanford University

March 17, 2026

Abstract

In a secret sharing scheme for a monotone access structure $\mathcal{A}$, one can share a secret among a set of parties such that all subsets of parties authorized by $\mathcal{A}$ can reconstruct the secret while all other subsets learn nothing. However, what if an unauthorized subset of parties collude and offer their shares for sale? Specifically, suppose that the parties pool their shares to create a reconstruction box that reveals the secret upon receiving enough additional shares as input. To deter this behavior, Goyal et al. (CRYPTO'21) introduced the notion of *traceable secret sharing* (TSS), where it is possible to provably trace reconstruction boxes containing leaked secret shares back to their respective parties. Goyal et al. and subsequent work presented definitions and constructions of TSS for the threshold access structure.

In this work, we revisit the notion of TSS.

- We identify shortcomings in previous formulations of TSS and present new, strengthened definitions that are not achieved by known constructions. We show that it is easy to build reconstruction boxes for which the tracing guarantees of *all* previous works fail.
- We extend the study of TSS beyond threshold to *general* access structures. Our new definitions are, in fact, necessary for this setting as natural adaptations of existing definitions become vacuous for general access structures.
- We present new constructions of TSS that satisfy our definitions for all access structures captured by monotone circuits. One of our constructions relies solely on one-way functions while the other additionally relies on indistinguishability obfuscation.

1 Introduction

In a secret sharing scheme [8, 52] for a monotone access structure $\mathcal{A}$, a dealer can share a secret s among a set of parties such that all authorized subsets of parties $A \in \mathcal{A}$ can recover s while all other subsets learn nothing about s. Secret sharing is a fundamental cryptographic primitive with many applications, most notably to secure multiparty computation [4, 16, 29] and threshold cryptography [18, 19].

Traceable Secret Sharing. Suppose that a subset of parties collude and sell the shares in their possession to whoever is willing to meet their price. In most secret sharing constructions, there is no way to hold such parties accountable, meaning that they have a risk-free incentive to sell the secret information.

To address this issue, Goyal et al. [34] introduced the notion of *traceable secret sharing* (TSS). Subsequently, new definitions were introduced by Boneh et al. [11], and investigated by several follow-up works. The basic idea is as follows. Suppose that an unauthorized subset of parties collude and pool their shares to create a *reconstruction box*. This box takes as input additional secret shares, and outputs the secret if it receives "enough" shares.[1] One may think of the reconstruction box as either an obfuscated program or a tamper-proof hardware device. The main requirement in a TSS scheme is *traceability*: There exists

[1] What constitutes 'enough' shares is specified by the instruction manual accompanying the box; we discuss this later.

an efficient tracer algorithm that can provably identify the parties responsible for constructing the reconstruction box given only black-box access to the box. To protect against malicious tracers, a TSS scheme must also satisfy *tracer non-imputability*, namely, even a malicious tracer should not be able to generate a false proof implicating an honest party, and *tracer secrecy*, namely, the secret remains hidden from the tracer even if it colludes with any unauthorized subset of parties.

Prior works [11, 34] formalized TSS for the threshold access structure and presented various constructions. Recently, [24] investigated TSS for general access structures in a *non-standard* model where the access structure is assumed to be hidden from the adversary (see Section 1.2 for details).

This Work. In this work, we revisit the notion of traceable secret sharing. First, we identify shortcomings in prior formulations of TSS and propose new and strengthened definitions that address these gaps. In particular, our definitions capture stronger traceability guarantees that are not achieved by known constructions. Second, we generalize the notion of TSS beyond threshold to general access structures. Finally, we give new constructions of TSS for all monotone access structures described by monotone circuits.

Before describing our results, we first provide an overview of the main themes of our work.

I. How to Model Traceability?. The notion of traceability is centered on the idea of a reconstruction box constructed by a subset of parties. Like any program or device, a reconstruction box must come with a user manual. For brevity, we refer to such a manual as the box *label*. In prior work, this label simply corresponds to a number $f < t$, where t is the reconstruction threshold. The box is guaranteed to work correctly if it receives a *good* set of shares where a set $I \subset [n]$ is said to be good if it satisfies the following:

- **No overlap:** It contains no overlapping shares with the reconstruction box. That is, $I \cap J = \emptyset$, where $J \subset [n]$ is the set of shares used to create the reconstruction box.
- **Minimal:** I is *minimal* in the sense that it contains exactly f shares and $f + |J| = t$. That is, the total number of input and corrupted shares is equal to the reconstruction threshold t.

Traceability is defined for reconstruction boxes that work correctly for *all* good sets.

We now highlight the key shortcomings of the above definition. The first issue concerns with the usability of the box: It is not clear how someone can ever get such a reconstruction box to work. Indeed, someone who buys the box needs to source sufficient shares from parties outside the set J, without knowing J, to satisfy the no-overlap requirement. This may require a super-polynomial number of queries to the reconstruction box if J is large, limiting the usability of such boxes.

The second and more pressing issue concerns the guarantees provided by the traceability definition: It only captures a rather limited class of reconstruction boxes. In particular, it does not capture boxes that work correctly only for:

- *Non-minimal* input sets, i.e., sets I s.t. $|I| + |J| > t$.
- A few good sets as opposed to *all* good sets.

Let us focus on the minimal set requirement. Intuitively, this only captures "well-behaving" adversaries that ask for minimal shares that suffice for reconstruction and no more. But why would an adversary not require non-minimal input sets, especially if it could help evade tracing? In fact, as it turns out, existing tracing strategies completely *fail* if the reconstruction box requires even one "extra" share, that is, $f = t - |J| + 1$. Note that such a box is easy to build, yet, as we explain in Section 2.1, it evades the tracing guarantees of *all* previous works[2]. Furthermore, as we explain below, if we extend the above requirements to the setting of general access structures, the notion of traceability becomes vacuous as minimal sets may simply *not* exist.

[2]Concurrent and independent of our work, Goyal and Waters also made this observation [33].

In this work, we present new definitions of traceability that overcome these limitations and, in particular, are meaningful for general access structures.

II. General Access Structures. Prior work on TSS has focused on the threshold access structure, leaving open the problem of TSS for other structures. In this work, we fill this gap and extend the study of TSS to general access structures.

As we alluded to earlier, existing definitions do not extend in a meaningful way to general access structures. First, for access structures where authorized sets are not symmetric, share count is clearly not an appropriate way to label a box. Second, and more importantly, for general access structures, we must necessarily allow for reconstruction boxes that might require *non-minimal* input sets: For any monotone access structure $\mathcal{A}$ and any set J that is unauthorized with respect to $\mathcal{A}$, let I be any set such that $I \cup J$ is authorized. We say that I is minimal if no subsets of $I \cup J$ are authorized; otherwise, I is non-minimal. Now consider an access structure over four parties: $(1 \vee 2) \wedge (3 \vee 4)$. For $J = \{3, 4\}$, it is easy to see that for any choice of I, $I \cup J$ is "over-authorized"; i.e., *all* sets I are non-minimal. In fact, this is true for many access structures.

On a technical level, TSS for general access structures poses fundamentally new challenges. For one, we must necessarily trace in the over-authorized setting, which for reasons that we discuss later, is challenging and requires new techniques. Furthermore, known efficient secret sharing schemes for general access structures (see, e.g., [5,6,36,53]) allow unauthorized sets of parties to perform a "partial reconstruction" on their shares such that the resulting value holds no information about the identities of the parties involved in the reconstruction, but can still be used towards full reconstruction given additional shares. This is fundamentally at odds with traceability since an adversary could simply create a reconstruction box with a partially reconstructed secret; there is no way to trace corrupt parties using such a box. We elaborate on these challenges in Section 2.1.

III. Boxes that do not Output Full Secret. So far, we have only discussed traceability for boxes that output the entire secret. However, in scenarios where even partial information about the secret is valuable, the corrupt parties may construct a box that does not output the full secret; instead, it may only output partial information, say, a *single bit*. Is it possible to trace in this setting?

The work of [34] considers such boxes, but allows for *non-black-box* tracing where the tracer algorithm has knowledge of how the reconstruction box was constructed. In this work, we remove this restriction and define black-box tracing for such boxes. A key benefit of our new notion is that it can be used to generically compile other cryptographic primitives, such as secret-key encryption schemes, to their traceable counterparts in the one-time use setting.[3]

1.1 Our Results

In this work, we give new definitions and constructions of TSS for general access structures. All our results are in the computational setting where the secrecy property of secret sharing holds against computationally bounded adversaries. We summarize our contributions in the following.

New Definitions. We start by modeling "minimally-useful" reconstruction boxes, namely, boxes that are accompanied with a label specifying a set of party identities I, such that they work correctly upon receiving shares of parties in the set I. Roughly speaking, upon receiving correct inputs, the box outputs one bit of information about the secret with some non-negligible probability. We formulate this via a semantic-security-style game (following [34]), where the box can distinguish between shares of two secrets s_0, s_1.

This models all useful boxes since any reconstruction box must work correctly on at least *one* share set; otherwise it is not useful. This is in contrast to prior work that only considers boxes that are guaranteed

[3] Achieving multi-use security would require a form of *reusable* secret sharing [20]. We leave exploration of this direction to future work.

to work correctly on *all* good input sets (as per the definition of good sets described earlier). We formalize two notions of black-box traceability for minimally-useful boxes:

- **$\emptyset$-Strong Traceability:** In this notion, we consider reconstruction boxes where I and J have no overlapping parties, i.e., $I \cap J = \emptyset$, where J is the set of corrupt parties whose shares were used to construct the box. We place no other restrictions on the box label; in particular, I can be non-minimal w.r.t. J.
- **Strong Traceability:** This notion is the same as above, except that we remove the no-overlap requirement. That is, we allow $I \cap J \neq \emptyset$.

We emphasize that *both* of these notions are stronger than existing notions of traceability. Following prior work, we also model security against malicious tracers by defining tracer non-imputability and tracer secrecy.

We are now ready to state our results.

I. TSS with $\emptyset$-Strong Traceability. Our first result is a TSS scheme for all access structures described by monotone circuits that achieves $\emptyset$-Strong Traceability as well as tracer non-imputability and tracer secrecy. Our scheme relies on the sub-exponential security of indistinguishability obfuscation (iO) [3,26,38,39] and one-way functions. To minimize the share size, our scheme is described in the public information model [1], where a public share is available to each party in addition to its private share.

Theorem 1.1 (Informal)**.** *Under the sub-exponential hardness of iO and one-way functions, there exists a TSS scheme for access structures described by monotone circuits, that achieves $\emptyset$-Strong traceability, tracer secrecy and tracer non-imputability in the public information model. For an access structure with monotone circuit C and n parties, public information is of size* $\mathsf{poly}(\lambda, n, |C|)$ *and private share size is* $\mathsf{poly}(\lambda, \log n)$*, where λ is the security parameter.*

We also show that in the case of a single dealer sharing multiple secrets, the public information can be re-used. Whenever the dealer wants to share a new secret, it only needs to publish a short public share whose size is independent of n and the circuit size (see Section 5 for details).

II. TSS with Strong Traceability. Next, we investigate the stronger notion of strong traceability. We first show that it is *impossible* to simultaneously achieve strong traceability and security against malicious tracers.

Theorem 1.2 (Informal)**.** *There does not exist TSS for threshold access structures and large secrets with strong traceability that also satisfies either tracer non-imputability or tracer secrecy.*

While strong traceability, in isolation, is a stronger property than $\emptyset$-strong traceability, the above result shows that TSS with $\emptyset$-strong traceability, tracer secrecy and tracer non-imputability is *incomparable* to TSS with strong traceability without tracer secrecy and tracer non-imputability (we will use TSS with strong traceability to refer to this notion from now on). In fact, we are able to construct TSS with strong traceability for monotone circuits relying solely on one-way functions. We obtain our result by devising a general compiler based on one-way functions from any secret sharing scheme for a monotone access structure $\mathcal{A}$ to a strongly-traceable TSS scheme for the same access structure. Our compiler preserves the share size of the underlying scheme up to a multiplicative factor in the number of parties n.

Theorem 1.3 (Informal)**.** *Assuming one-way functions, if there exists an efficient secret sharing scheme* SS *for a monotone access structure $\mathcal{A}$, then there exists a strongly-traceable TSS scheme for the same access structure $\mathcal{A}$ with share size n times the share size of* SS*.*

By applying our compiler to Yao's secret sharing scheme for monotone circuits [53], we obtain TSS with strong traceability for monotone circuits based on one-way functions. Finally, we show that we can also extend our scheme in Theorem 1.1 to achieve strong traceability at the cost of tracer secrecy and tracer non-imputability, while retaining the same share size (see Appendix B).

Application to One-time Traceable Secret-key Cryptography. Our constructions of traceable secret sharing (TSS) can be used to generically obtain one-time traceable secret-key encryption for general access structures.

We introduce the notion of $\emptyset$-strong traceability in the encryption setting, defined with respect to a minimally-useful decryption box labeled by a set I and a ciphertext. This box takes as input decryption shares of parties in I, and outputs decryption of the ciphertext. We then show how to compile any one-time secret-key encryption scheme to a traceable scheme for general access structures, by simply secret-sharing the decryption key using our TSS. Crucially, since our TSS schemes achieve black-box tracing for boxes that only output partial information about the secret, we can trace boxes that only output the decryption of a ciphertext (and not the reconstructed decryption key). Indeed, this compiler cannot be realized using prior TSS constructions (even for the threshold access structure), since [11] only support tracing boxes that output the full secret and [34] do not achieve black-box tracing. We present the full compiler in Appendix C.

Open Problems. Our work leaves open several interesting questions for future research. First, determining minimal assumptions for achieving our new, strengthened notions of TSS remains open. While the use of obfuscation seems crucial to the design of our first TSS scheme (see Section 2), it is not clear whether obfuscation or even public-key cryptography is necessary. This question remains open even for the case of threshold access structures.

The second question concerns with share-size efficiency of TSS. Our first scheme has succinct private shares, but requires a large public share whose size grows with the size of the access structure. Minimizing the size of the public share will likely require building upon recent advances in succinct secret sharing [37, 45]; we leave exploration of this direction to future work.

1.2 Related Work

Traceable Secret Sharing. The notion of traceable secret sharing was introduced by Goyal et al. [34], who proposed a construction for threshold access structures based on the Goldreich-Levin decoding algorithm [28]. Subsequently, Boneh et al. [11] revisited this concept and introduced an alternative traceability model, presenting two new constructions for threshold access structures based on the schemes of Shamir [52] and Blakley [8]. Hoffmann [35] proposed a construction for threshold access structures using Mignotte's scheme [46] albeit only achieving weak (inverse polynomial) secrecy. Baghery et al. [2] designed a traceable verifiable secret sharing scheme for threshold access structures.

Recently, Farràs et al. [24] investigated TSS for general monotone access structures in a non-standard model where the access structure is assumed to be hidden from the adversary. If this assumption is violated, their tracing guarantee does not hold. In this work, we consider the standard model and do not make such assumptions on the adversary. Beyond this distinction, there are additional differences between their work and ours. Similarly to prior work, their traceability definition suffers from the limitations discussed in Section 1. Their scheme is linear and achieves information-theoretic security, but requires share size and reconstruction time exponential in the number of parties. Our constructions, in contrast, are efficient but only achieve computational security.

In a concurrent work, [44] construct a TSS scheme for threshold access structures based on Asmuth-Blum secret sharing. Their definitions suffer from the same limitations as described in Section 1.

Anonymous Secret Sharing. Anonymous secret sharing requires that shares from an unauthorized sub-

set of parties do not reveal any information about party identities or even that the shares were generated together. We refer the reader to [7] for a survey of this line of work. We note that the partial reconstruction challenge that arises in the design of TSS (see Section 2) is also a barrier to achieving anonymity. In particular, to achieve anonymity, one must ensure that any attempt at partial reconstruction does not reveal that the shares were generated together. We note, however, that hiding such correlations is *not* necessary for TSS. Indeed, unlike the work of [7], we are able to address the partial reconstruction problem – in the setting of strong traceability – using only one-way functions.

Secret Sharing with Snitching. The goal of holding secret-share holders accountable has inspired another line of work called secret sharing with snitching [14,23]. These works consider a different adversarial model wherein if an authorized set of parties collude to reconstruct the secret, then one of the colluding parties learns a "snitching proof" indicating that some party participated in illegal secret reconstruction.

Traitor Tracing. In the traitor tracing setting, a central authority issues a single encryption key, and n decryption keys, one per receiver. If any coalition of receivers comes together and produces a pirate decoder box D that can decrypt well-formed ciphertexts, then this box can be traced back to at least one of the receivers who contributed to it. Many traitor tracing schemes have been proposed over the years (see, e.g., [9,13,15,17,21,22,25,27,30,31,41,43,47,48,50,54,55]). As we explain in Section 2, traitor tracing techniques can be used to build TSS for the star graph access structures, but do not suffice for general access structures.

2 Technical Overview

In this section, we describe the main ideas that underlie our results. We start by providing an overview of the main challenges towards achieving our notions of traceability. We describe our construction that achieves $\emptyset$-strong traceability in Section 2.2. Finally, we discuss the case of strong traceability in Section 2.3.

2.1 Main Challenges

In the following, we discuss two key challenges that we must overcome to realize our notions of TSS.

Challenge I: Over-authorization. Recall that in our traceability definitions, a reconstruction box built using shares of parties in J may require input shares of sets I s.t. $I \cup J$ is over-authorized, i.e, there are multiple ways to reconstruct the secret using shares of $I \cup J$. For example, let us consider the t-out-of-n threshold access structure. Suppose that J consists of f parties and the box requires input sets I consisting of $t - f + 1$ shares different from J. In this case, the box has $t + 1$ different ways to reconstruct the secret.

Known tracing algorithms [11,34] run the reconstruction box with "fake" shares that despite not being "real", are consistent with adversarial shares. For example, in the TSS scheme of [11] based on Shamir secret sharing, each share consists of an evaluation of a random degree $(t - 1)$ polynomial (with the constant term set to the secret) on a random point. The tracer algorithm in this scheme prepares the shares of an input set I by sampling random field elements. When $|I| = t - f$, fake shares are indistinguishable from real shares to the reconstruction box since a randomly sampled set of $t - f$ points forms a unique degree $t - 1$ polynomial when combined with the f shares in the box.

However, this strategy *fails* when $|I| > t - f$. Note that in this over-authorized setting, the box may run the reconstruction algorithm using different authorized subsets of $I \cup J$ and abort if the answers do not match. Unfortunately, in this setting, there is no way to sample shares consistent with adversarial shares without knowledge of real shares. For example, let $f = t - 1$ and $|I| = t - f + 1 = 2$. Then, each of the two different shares sampled by the tracer will form a different degree $t - 1$ polynomial with the adversarial

shares. Indeed, for this reason, existing tracing strategies cannot work in the over-authorization setting. This necessitates new tracing techniques.

Challenge II: Partial Reconstruction. The standard methodology for constructing efficient secret sharing for general access structures (see, e.g., [5, 6, 36, 53]) involves a recursive design: To share a secret s for an access structure, say $\mathcal{A} = \mathcal{B} \wedge \mathcal{C}$, we first share s into $s_{\mathcal{B}}$ and $s_{\mathcal{C}}$ for the $\wedge$ gate and then recursively share $s_{\mathcal{B}}$ (resp., $s_{\mathcal{C}}$) w.r.t. $\mathcal{B}$ (resp., $\mathcal{C}$). For example, consider the access structure $(1 \vee 2) \wedge (3 \vee 4)$. Here, secret s is first additively shared into s_ℓ and s_r, and parties 1 and 2 are assigned s_ℓ as their share. Similarly, parties 3 and 4 are assigned s_r.

Now suppose that party 1 is corrupt and uses its share s_ℓ to construct a reconstruction box. Since this box could also have been constructed by the party 2 (since they are both authorized for the left gate $\vee$), there is no way to definitively trace the box to party 1. More generally, for any sub-access structure G within an access structure $\mathcal{A}$, a set of parties that are authorized w.r.t. G could use their shares to reconstruct the value s_G associated with the output wire of G. If there are multiple subsets of parties that are authorized w.r.t. G, then s_G contains no information about the reconstructing set of parties. Thus, if the corrupt parties create a reconstruction box using s_G, there is no way to trace their identities.

Note that this issue does not arise in the case of Shamir secret sharing for threshold access structure (for threshold greater than one). To see why, consider a reconstruction box built by a set of parties J, that takes as input shares of a set I. Although some partial reconstruction is possible by combining the shares of J using Lagrange coefficients for the set $J \cup I$, the box will still have to store some information about the set J so that it can compute Lagrange coefficients for the input shares of I. Indeed, this is why prior work that focused on the threshold access structure does not need to separately tackle this issue. For the case of general access structures, however, new ideas are required to address this.

We now proceed to describe the main ideas underlying our constructions, starting with our construction of TSS with $\emptyset$-Strong Traceability.

2.2 TSS with $\emptyset$-Strong Traceability

Our Strategy in a Nutshell. We start by giving a bird's eye view of how we address the challenges discussed above, and then provide more details.

Recall that in order to achieve our traceability definitions, we must address the over-authorization challenge: The tracer must seemingly provide shares for parties in the set I as input to the reconstruction box, while ensuring consistency with the shares of corrupt parties J—even when $I \cup J$ forms an over-authorized set. At first, this seems impossible; indeed, as discussed earlier, existing tracing techniques [11, 34] fundamentally break down in this setting.

To address this challenge, we eliminate the tracer's burden of preparing consistent shares. Instead, we design a scheme where any inconsistencies in the shares are *invisible* to the adversary. We design our scheme in the public information model, where an additional public share is essential for reconstruction. This share will enable two *indistinguishable* modes of reconstruction:

- Normal mode, where reconstruction succeeds only with consistent shares.
- *Tracer* mode, where it is possible to *program* the output of reconstruction using shares generated by the tracer.

Since the two modes are indistinguishable, the adversary cannot tell whether the output comes from honest reconstruction or from the tracer's programmed shares. Thus, even if the box attempts reconstruction using different authorized subsets of $I \cup J$, it will always see the same output programmed by the tracer-generated shares of I (even if these shares are inconsistent with adversarial shares). We implement this blueprint

using indistinguishability obfuscation (iO). Roughly speaking, the public share will be an obfuscation of a modified reconstruction algorithm with two indistinguishable modes of operation.

To address the partial reconstruction challenge, we implement an *all-or-nothing* design: The reconstruction algorithm reveals *nothing* unless the input shares form an authorized set. In other words, partial reconstruction is not possible. Technically, we achieve this by programming the obfuscated reconstruction algorithm to abort whenever invoked on shares corresponding to an unauthorized subset. Later, in Section 2.3, we design a different mechanism to implement an all-or-nothing strategy using only one-way functions in the setting where we do not require tracer secrecy.

Warm-up: Tracing by Observing "Jumps". Before describing our scheme, we describe a tracing technique that we will use in our constructions. This tracing technique underlies many works on traceable encryption for the 1-out-of-n access structure (commonly referred to as traitor tracing) [12, 17, 27, 30, 31, 54–56].

To explain this technique, we consider a warm-up task of constructing TSS for the star graph access structure on $n+1$ parties, where an authorized set consists of the "star" party $n+1$ together with any one of the "leaf" parties $1, \ldots, n$. A simple secret sharing scheme for this access structure can be constructed as follows: We additively share the secret s as $s = s_{n+1} + s_{[n]}$, giving s_{n+1} to the star party $n+1$ and $s_{[n]}$ to each of the leaf parties. Now consider an adversary who corrupts a subset $J \subseteq [n]$ and constructs a reconstruction box labeled with $I = \{n+1\}$. In this case, since the box has no information about which party out of $\{1, \ldots, n\}$ contributed to the box, tracing is impossible.

Fortunately, we can address this issue by constructing a different secret sharing scheme using a special encryption scheme called Private Linear Broadcast Encryption (PLBE) [12]. An n-party PLBE scheme achieves the following special properties:

- Any normal (i.e., honestly generated) ciphertext can be decrypted by any of the n parties, where each party holds a *distinct* decryption key.
- It is possible to create "special" ciphertexts that can only be decrypted by a subset of the parties, where the subset is specified by a contiguous range (i, n) for $i \in [n]$. For convenience, we refer to this range as the ciphertext *tag*. It follows that a special ciphertext with tag $(1, n)$ is the same as a normal ciphertext.
- The special ciphertexts satisfy *index-hiding*, which means that two special ciphertexts with tags (i, n) and $(i+1, n)$ are indistinguishable to an adversary holding keys for all parties $\ell \neq i$.

Intuitively, special ciphertexts allow for *selective disabling* of the keys of some parties. Crucially, the ciphertext reveals minimal information about which parties have been disabled: A party that is allowed to decrypt will learn that it is indeed not disabled, but nothing more.

Given a PLBE scheme, we can construct a TSS for the star graph as follows. To share a secret s, we create a normal encryption of s and set it as the share of the star party. Each leaf party in $\{1, \ldots, n\}$ receives a distinct decryption key. For tracing, recall that we are considering a reconstruction box that requires as input the share of party $n+1$, which is a ciphertext. The tracer is given as input an encryption key that allows it to create special ciphertexts. Using this key, the tracer runs the box many times with different ciphertexts, wherein it disables each party one by one. Specifically, it creates the share of party $n+1$ with a special ciphertext for set $\{1, \ldots, n\}$, then $\{2, \ldots, n\}$, and so on until $\emptyset$.

When the box is run with a ciphertext for $\{1, \ldots, n\}$ that can be decrypted by any party (like a normal ciphertext), a useful box must be able to decrypt. On the other hand, given a ciphertext such that all the parties are disabled, i.e., for the set $\emptyset$, the box cannot decrypt. This means that there must be a *jump* in the behavior of the box. Concretely, there must exist a party $j^* \in [n]$ such that when the tracer disables this party by encrypting to set $\{j^*+1, \ldots, n\}$, the box will suddenly stop working, or at the very least,

the probability with which the box can reconstruct the secret (relative to a ciphertext where this party is not disabled) will noticeably reduce. The tracer blames this party j^*. Correctness of this strategy hinges on the index-hiding property: without the key of j^*, the adversary cannot tell whether j^* was disabled or not. Thus, the jump in behavior correctly implicates j^*.

Our First Attempt. Armed with this jump-based tracing technique, we now proceed to implement our blueprint for constructing TSS with $\emptyset$-strong traceability. Initially, we will only focus on achieving traceability. Later, we will extend our solution to also achieve security against malicious tracers.

Our construction relies on a symmetric-key encryption scheme $\mathsf{SKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and a signature scheme $\mathsf{Sig} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Looking ahead, we will, in fact, require stronger tools even for achieving standard secrecy: Specifically, a *puncturable* encryption scheme and a *constrained* signature scheme. However, for now, we ignore these details. We share a secret s as follows:

- We sample keys for the encryption and signature schemes: $\mathsf{sk}_{\mathsf{SKE}} \leftarrow\$ \mathsf{SKE.KeyGen}(1^\lambda)$ and $\mathsf{sk}, \mathsf{vk} \leftarrow\$ \mathsf{Sig.KeyGen}(1^\lambda)$.
- For each party $i \in [n]$, the share contains an encryption to zero, i.e., $\mathsf{ct}_i \leftarrow\$ \mathsf{SKE.Enc}(\mathsf{sk}_{\mathsf{SKE}}, 0)$. Roughly speaking, this is an encryption with respect to the set $\{1, \ldots, n\}$, i.e., no party is disabled (note that there is no message being encrypted here). In general, we will encrypt to a set $\{j+1, \ldots, n\}$ by simply encrypting j using SKE.
- The share of each party additionally contains a signature on the party's index and ciphertext, i.e., $\sigma_i \leftarrow\$ \mathsf{Sig.Sign}(\mathsf{sk}, (i, \mathsf{ct}_i))$. We set $\mathsf{sh}_i := (\mathsf{ct}_i, \sigma_i)$ for all parties. Looking ahead, the signature will act as an authentication key.
- We compute the obfuscation of a program P. For now, let us assume that we have virtual black-box (VBB) obfuscation [3]. This program P has the secret s, the circuit C representing the access structure, the symmetric key $\mathsf{sk}_{\mathsf{SKE}}$ and the signature verification key vk hard-coded. It takes as input the shares of a subset of parties. Figure 1 gives an overview of this program – at a high level, it outputs the secret if and only if, the subset of input parties that remain enabled with respect to the ciphertexts in their shares is still authorized under C. Otherwise, it simply outputs $\perp$. This obfuscated program serves as the public share of our scheme.

Hard-coded inputs: Secret s, Access structure C, Signature verification key vk, Decryption key $\mathsf{sk}_{\mathsf{SKE}}$.

1. For each input share $\mathsf{sh}_i = (\mathsf{ct}_i, \sigma_i)$:
   (a) Verify the signature, i.e., ignore this share if $\mathsf{Sig.Verify}(\mathsf{vk}, \sigma_i, (i, \mathsf{ct}_i)) = 0$.
   (b) Decrypt the ciphertext to get an index, i.e., $j_i \leftarrow \mathsf{SKE.Dec}(\mathsf{sk}_{\mathsf{SKE}}, \mathsf{ct}_i)$.
   (c) Ignore shares of all disabled parties, i.e., all parties $\leq j_i$.
2. Output the secret s if the remaining set of parties is still authorized. Otherwise, output $\perp$.

Figure 1: An overview of the reconstruction algorithm for our first attempt to achieve $\emptyset$-Strong traceability. An obfuscation of this algorithm forms the public information.

To reconstruct the secret, an authorized set of parties simply runs the obfuscated program in the public share, with their shares as input. Correctness follows because the share of each party contains an encryption of 0, which corresponds to the set $\{1, \ldots, n\}$. Hence, when run with the shares of any authorized set,

the program will not ignore any party, meaning that it will always output the secret. In contrast, when invoked on any unauthorized set, the program returns $\perp$, thereby preventing partial reconstruction.

We now turn to traceability, postponing the discussion of secrecy until later in this section. Our goal is to incorporate the jump-based tracing approach described earlier. Recall that the tracer gets as input the label I such that the reconstruction box only works when given shares of parties in I. At a high level, tracing proceeds by repeatedly running the box on different tracer-generated shares of I, where the ciphertexts in those shares progressively disable parties one by one. Concretely, the tracer prepares shares of I with ciphertexts corresponding to sets $\{1, \ldots, n\}$, $\{2, \ldots, n\}$, and so on till $\emptyset$. To enable the tracer to prepare such ciphertexts as well as signatures on these ciphertexts, the encryption and signing keys , i.e., $\mathsf{sk}_{\mathsf{SKE}}$ and sk are included in the tracing key.

By the same reasoning as in the standard jump-based approach, when the box is run with shares containing ciphertexts that enable all parties (like real shares), a useful box must output information about the secret. On the other hand, when run with shares containing ciphertexts that disable all parties, i.e., for the set $\emptyset$, the box cannot work. Intuitively, this is because, all the box can do is run the obfuscated program on shares of authorized subsets of $I \cup J$, since the box does not have a valid authentication key, i.e., signature for any party outside of this set. Next, since J is unauthorized by definition, any authorized subset of $I \cup J$ must contain at least one party in I. This means that, if the box runs the obfuscated program for any authorized subset of $I \cup J$, the program will never output the secret because the ciphertexts in the tracer-generated shares of I disable all parties. Hence, there must exist an index j^* such that the output behavior of the box changes by a noticeable amount when the tracer disables this index. It remains to show that this jump in behavior cannot occur at an honest party.

Intuitively, since no party has access to the decryption key $\mathsf{sk}_{\mathsf{SKE}}$, the only way for the adversary to distinguish whether a party j is enabled by the share of a party i is to compare the outputs of the obfuscated program on $i \cup S$ versus $i \cup S \cup j$ for some set $S \subseteq [n] \setminus \{i, j\}$. However, producing the latter input requires a valid share for party j, which in turn requires forging a signature. Hence, an adversary without the real share of party j^* cannot tell whether this party is allowed by the ciphertexts in shares of I. The tracer therefore safely accuses j^* as corrupt. This first attempt can be viewed as an adaptation of the iO-based PLBE framework of [13].

Insufficiency of PLBE. Unfortunately, there is a flaw in the above argument. While the adversary only has the authentication keys, i.e., signatures for the corrupt parties J, the reconstruction box receives as input the tracer-generated shares of I. These shares contain ciphertexts as well as valid authentication keys for all parties in I. Consequently, the adversary effectively has access to authentication keys for I, and may be able to distinguish whether a party $j^* \in I$ has been disabled. In this case, the box could cause a jump just at index $j^* \in I$ and nowhere else, meaning that the tracer will fail to identify any corrupt party! Therefore, to achieve $\emptyset$-strong traceability, we need to strengthen the jump-based tracing technique.

Private *Set* Broadcast Encryption. At a high level, our solution is to never disable the parties in I during the tracing process. To achieve this, we generalize the PLBE primitive so that we can encrypt to sets of the form $\{1, \ldots, n\} \cup I$, $\{2, \ldots, n\} \cup I$ and so on till $\emptyset \cup I$ for an arbitrary set $I \subseteq [n]$. We refer to this as *private set broadcast encryption* – a generalization of private linear broadcast encryption. Given such a scheme, we can now modify the jump-based tracing strategy as follows: the tracer prepares shares of label I with ciphertexts for $\{1, \ldots, n\} \cup I$, $\{2, \ldots, n\} \cup I$ and so on till $\emptyset \cup I$. It then identifies the index j^* at which the box's behavior exhibits a jump, and blames this party.

Let us now see why this private set broadcast encryption based approach solves the problem. As in our first attempt, a useful box must output information about the secret when provided shares with ciphertexts for $\{1, \ldots, n\} \cup I$. On the other hand, the box cannot work when given shares containing encryptions for I, which disable all parties outside I. Indeed, note first that the label I must not be an authorized set – a box that asks for shares of an authorized set as input is not useful, since one could simply reconstruct the

secret using shares of I. Then, by a similar argument as above, the obfuscated program will never output the secret when run with any authorized subset of $I \cup J$ (which must contain some party in I), because the only set of parties allowed by the ciphertexts in shares of I, is the set I, which is unauthorized to begin with. Thus, there must be an index j^* such that the behavior of the box changes when the tracer disables this index. This index cannot be in $[n] \setminus (I \cup J)$, as argued for our first attempt. But crucially, this jump cannot be in I either, since parties in I are never disabled. Hence, the tracer is guaranteed to always find a corrupt party.

To implement this private set broadcast encryption based strategy, we need to modify the ciphertexts contained in each share so that the tracer can encrypt to sets of the form $\{j, \ldots, n\} \cup I$, when preparing shares of the label set I. A natural first attempt is to simply use symmetric key encryption to encrypt both the set I and the index j, and modify the obfuscated program to only allow parties in $\{j+1, \ldots, n\} \cup I$, when it is given such a ciphertext. While correct, this approach leads to ciphertexts of size $|I|$, which can grow with $\Theta(n)$.

To avoid this blow-up in the share size, we exploit the fact that the tracer does not need to embed the set I in the ciphertext for *every* party's share in I. Instead, we modify the encryption in each ciphertext to encrypt a one-bit flag f denoting whether this share is tracer-generated, together with an index j (as before). Specifically, each party's share will now include an encryption of $f = 0, j = 0$ (along with a signature, as before). During tracing, the tracer will set the flag to 1 for all the shares that it prepares, and varies the index j from 0 to n, as before. The obfuscated program is modified so that it never ignores shares marked with $f = 1$. Figure 2 depicts the modified obfuscated program. As discussed above, the tracer will find at least one jump, and this jump must be corrupt. Crucially, the share size now grows only poly-logarithmically in the number of parties.

Hard-coded inputs: Secret s, Access structure C, Signature verification key vk, Decryption key $\mathsf{sk}_{\mathsf{SKE}}$.

1. For each input share $\mathsf{sh}_i = (\mathsf{ct}_i, \sigma_i)$:
   (a) Verify the signature, i.e., ignore this share if $\mathsf{Sig.Verify}(\mathsf{vk}, \sigma_i, (i, \mathsf{ct}_i)) = 0$.
   (b) Decrypt the ciphertext to get a flag and an index, i.e., $f_i, j_i \leftarrow \mathsf{SKE.Dec}(\mathsf{sk}_{\mathsf{SKE}}, \mathsf{ct}_i)$.
   (c) Ignore shares of all disabled parties $\leq j_i$ without the flag, i.e., ignore all j such that $f_j = 0$ and $j \leq j_i$.
2. Output the secret only if the remaining set is still authorized. Otherwise, output $\perp$.

Figure 2: An overview of the reconstruction algorithm for our TSS scheme that achieves $\emptyset$-Strong traceability. The public information contains an obfuscation of this algorithm.

Achieving Secrecy. So far, we assumed VBB obfuscation, under which secrecy is straight-forward. Since VBB obfuscation is impossible in general, we instead use iO to obfuscate the program in our public share. To argue secrecy with iO, we proceed via a sequence of hybrid experiments in which the secret is gradually removed from the obfuscated program. We then need to argue that an adversary with the shares of an unauthorized set $J \subset [n]$ cannot tell that the secret has been removed. This requires that the obfuscated program's functionality remains unchanged for such an adversary. In particular, we must ensure that no valid signatures exist for parties outside the corrupt set J. With this property, the program ignores all shares outside J, and since J is unauthorized, it never outputs the secret—allowing us to argue that removing the secret does not alter observable behavior. To achieve this, we need to use a *constrained signature scheme*, which allows generating a *constrained verification key* vk^* under which *no valid signatures*

exist for messages satisfying a given constraint. Similarly, to argue traceability in the iO setting, we need to instantiate the symmetric-key encryption using a puncturable PRF.

Achieving Tracer Secrecy via *Doubly* Constrained Signatures. A careful reader might observe that the scheme does not achieve tracer secrecy. Recall that the tracing key contains both the decryption key $\mathsf{sk}_{\mathsf{SKE}}$ and the signing key sk, to allow the tracer to prepare shares for tracing. This means that the tracer can simply generate fresh, valid shares for all parties by generating encryptions to flag 0 and index 0, along with valid signatures, and run the obfuscated program with these shares to learn the secret.

To prevent this, we need to restrict the tracer's power so that it cannot generate real shares of any party; at the same time, it must still be able to trace. We observe that for tracing, it suffices that the tracer can sign ciphertexts encrypting flag $f = 1$ and arbitrary indices j—it does not need to sign ciphertexts with $f = 0$. Hence, we define a new type of signature scheme that we call *doubly-constrained signature*, wherein we can generate a *constrained* signing key, which can only be used to sign messages that satisfy a certain constraint. We call this doubly-constrained since it also allows generating a constrained verification key (as defined above), such that valid signatures only exist on messages that satisfy a certain constraint. We now use this scheme to sign the ciphertext in each share. The tracer is given a constrained signing key as part of the tracing key, which can only sign ciphertexts that encrypt the flag set, i.e., $f = 1$, ensuring that the tracer cannot generate valid, real shares with the flag $f = 0$.

However, the obfuscated program in Fig. 2 would still output the real secret even if the inputs contain tracer-generated shares with flag $f = 1$. To ensure tracer secrecy, we also need to modify the obfuscated program to never output the real secret if it sees any such tracer-generated share. But, what should it output instead? Clearly, the program must output a value that is identically distributed as the secret (or at least be computationally indistinguishable from the secret's distribution), as otherwise the reconstruction box can trivially distinguish between real shares and tracer-generated shares. To solve this, we observe that the tracer does indeed know the distribution of the secret. Recall that the reconstruction box can distinguish between shares of two secrets s_0, s_1, which are also known to the tracer. This means that the real secret is distributed uniformly at random between these two values. Hence, if we could *program* the output of the obfuscated program, then we can force it to output something that comes from the same distribution as the secret, when given tracer-generated shares as input.

We achieve this by modifying the ciphertext in each share to now encrypt three values, a flag f used to indicate tracer-generated shares, an index j used for tracing, and a value m. This value is set to 0 for real shares of all parties. But during tracing, the tracer samples a value s^* from the same distribution as the secret – it simply picks one of s_0, s_1 uniformly at random. Then, when preparing shares for all parties in I, it generates ciphertexts encrypting s^* as the value (along with flag $f = 1$ and index j ranging from 0 to n). We modify the obfuscated program to simply output this value when it sees any share with the flag set, i.e., $f = 1$[4]. This allows us to achieve tracer secrecy; roughly speaking, the tracer can only generate shares encrypting the flag set, and the obfuscated program when run with such shares, never outputs the real secret. Combining this with the tracing technique described above, we have constructed a TSS with both $\emptyset$-Strong traceability and tracer secrecy.

In Section D, we present a construction of doubly constrained signatures from sub-exponential hardness of iO. In Section 5, we give a formal description and security analysis of our TSS construction.

Achieving Tracer Non-imputability. Recall that to achieve security against a malicious tracer, we also need tracer non-imputability, meaning that a malicious tracer must not be able to blame an honest party. In the scheme described so far, there is nothing stopping a malicious tracer from just blaming any honest party i, by simply outputting the index i. To achieve tracer non-imputability, informally, there must be some secret value in the share of each party, which is only known to this party; and if the tracer can extract this secret value for some party from the reconstruction box, then it can be used as proof for blaming that

[4]This can be seen as the *tracer* mode of the obfuscated program, that we alluded to in the beginning of this section.

party. Concretely, we assign a random identifier $x_i \in \{0,1\}^\lambda$ from a large space to each party $i \in [n]$. We also include a one-way function of this identifier $\mathsf{OWF}(x_i)$ in the tracing and verification key. Now, the tracer must recover this identifier to blame a party i. Intuitively, falsely accusing an honest party i would amount to inverting $\mathsf{OWF}(x_i)$ for a randomly chosen x_i.

But it is unclear how to trace now; the tracer not only needs to identify a corrupt party j, it must also extract the corresponding identifier x_j from the box. To do so, we observe that this problem is similar to a notion called embedded-identity traitor tracing [32, 49], wherein each party also has an *identity* (which may be a string) along with an index, and the tracer must extract the identity of a corrupt party along with its index. The identifier x_j in our scheme can be interpreted as the identity of that party.

Based on this connection, we adapt techniques from [32] to extract the identifier from a reconstruction box. At a high level, we generalize the encryption scheme so that we can disable parties based on not only their index and flag (as done above), but also on a single bit of their identifier. This will allow the tracer to extract the corrupt identifier one bit at a time. In more detail, the tracer does the following: First, it extracts the index j^* of a corrupt party by using the jumps approach as described above. Then, for each position of the identifier, it will prepare shares of parties in I with ciphertexts that disable party j^* if its identifier is 0 at that position (in addition to disabling parties $< j^*$ based on their flags). Roughly speaking, if the box can distinguish between these ciphertexts and the ciphertexts that only disable parties $< j^*$ based on their flags, then x_{j^*} must be 0 at that position (and 1 otherwise).

To implement this technique, we again use symmetric-key encryption to encrypt a position $u \in \{0, \ldots, \lambda\}$ and a bit $b \in \{0,1\}$ (along with a flag f, an index j, a value m as above). The share of party i will contain an encryption ct_i to position 0, bit 0 (and flag 0, index 0 and value $m = 0$ as above), and a signature, which now also signs the identifier x_i, along with the index i and ciphertext ct_i. We also modify the obfuscated program to take as input shares of the form $(i, x_i, \mathsf{ct}_i, \sigma_i)$ and interpret them as follows:

- Verify the signature and if valid, decrypt the ciphertext to get (u, b, f, j, m).
- If $u = 0$, the program disables all shares for parties $\leq j$, but keeps all shares with flag $f = 1$. This is essentially equivalent to an encryption to (f, j, m) as described above.
- If $u > 0$, then the program also disables $j + 1$ if the uth bit of the corresponding identifier x_{j+1} is b.

The rest of the program remains the same as before: it outputs $\perp$ if the remaining set of parties is not authorized; otherwise, it outputs the programmed value or the real secret, based on the flags in the shares. We give a formal description of our scheme with tracer non-imputability in Section 5.4.

2.3 TSS with Strong Traceability

Impossibility Result. We start by providing brief intuition on our impossibility result. Recall that in the definition of Strong traceability, we consider reconstruction boxes with very little restriction on the label I. In particular, the label may even contain corrupt parties, i.e., $I \cap J \neq \emptyset$. We present a negative result in Appendix A, which states that it is in fact impossible to achieve this notion of traceability simultaneously with tracer secrecy or tracer non-imputability, if the secret comes from a large space.

To understand why, consider a particular reconstruction box with a label I such that $I \cap J$ is indeed a non-empty set. This box simply aborts if the shares given as input do not match the real shares for parties in $I \cap J$; it can do this because the box already has the real shares of all corrupt parties in J. Hence, to get such a box to output something meaningful, the tracer will need to prepare shares of I such that the shares of all parties in $I \cap J$ are exactly the same as their real shares. On the other hand, if the tracer has the ability to prepare real shares for $I \cap J$, then it can also break tracer secrecy! It can simply corrupt an unauthorized set of parties S such that $S \cup (I \cap J)$ is authorized, then run the reconstruction algorithm

on shares of S and the shares of $I \cap J$ that it prepared. Since all of the prepared shares for $I \cap J$ are the real shares, the reconstruction algorithm will indeed output the secret by correctness.

Similarly, if the tracer has the ability to prepare real shares for $I \cap J$, then it can also break tracer non-imputability. It can simply prepare a reconstruction box with the shares of $I \cap J$, and run the Trace on this box. By strong traceability, we know that the Trace algorithm must find at least one party in $I \cap J$ along with a valid proof; hence allowing the tracer to blame this party without ever querying for their real share.

We now describe our construction of TSS with Strong traceability from one-way functions.

Starting Observations Towards Positive Result. To achieve Strong traceability, we leverage the impossibility result to our benefit. Given that there is no hope of achieving tracer secrecy, we can make the task of tracing easier by simply providing the real shares of all parties as part of the tracing key. The tracer can then use the real shares when preparing shares of the label I. This allows us to completely bypass the first challenge – the real shares are consistent with the adversarial shares by definition! Hence, we now only have to deal with the second challenge, i.e., partial reconstruction when designing a traceable scheme. While we used obfuscation to achieve this in the last section, we will only rely on one-way functions here. We start with describing our approach for tracing reconstruction boxes with label I that does not overlap with the corrupt set J, i.e., $I \cap J = \emptyset$; we will discuss how to remove this restriction later in the section.

Dealing with Partial Reconstruction. To get around this challenge, we need to take the *all-or-nothing* approach mentioned in Section 2.2 to design the reconstruction algorithm such that an unauthorized subset of parties cannot learn anything meaningful. Moreover, for our tracing via jumps approach to work, we still need the ability to create shares of parties such that we can disable shares of certain parties.

As a naive first attempt, let us encrypt the secret using any encryption scheme, and publish the resulting ciphertext as the public share. Then, we secret share the decryption key using any secret sharing scheme SS for monotone circuits [1, 45, 53], and give the shares of the key as secret shares to all parties. While this avoids partial reconstruction of the secret, a subset of parties can still partially reconstruct the key! We conclude that any one-level secret sharing approach (where we share the secret or a key hiding the secret) will not be enough to avoid partial reconstruction.

Our Approach. Instead, we devise a *two-level* sharing procedure to address the partial reconstruction challenge. At a high level, our approach is to first share the secret using any secret sharing scheme SS for monotone circuits [1, 45, 53], and then compute a *special* encryption of each share. This special encryption scheme is a generalization of private set broadcast encryption (which in turn, generalizes PLBE) to general access structures. It has three key properties:

- A normal ciphertext can only be decrypted if an authorized subset of parties comes together (as opposed to any single party being able to decrypt).
- It is possible to create "special" ciphertexts with respect to any set S as the tag, such that only authorized subsets of S are allowed to decrypt, essentially disabling all parties outside S.
- The special ciphertexts leak limited information about the tag S – parties in S can learn that they are not disabled, but nothing more. This generalizes the notion of index-hiding for PLBE.

Roughly speaking, each party will only get the special encryption of its share (along with the special decryption key corresponding to its index). The first property will ensure that, if an unauthorized subset of parties collude, they will not be able to decrypt any of the shares, hence no partial reconstruction would be possible. The second and third properties will allow us to use the tracing via jumps approach described in Section 2.2.

We now describe our TSS scheme, including how we implement the special encryption with these properties. We will use a secret-key encryption scheme $\mathsf{SKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and a secret sharing

scheme $\mathsf{SS} := (\mathsf{Share}, \mathsf{Rec})$ for monotone circuits as building blocks. To share a secret s with respect to a monotone circuit C as the access structure, we do the following:

- Share s using SS, i.e., $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n) \leftarrow\!\!\$\ \mathsf{SS.Share}(1^\lambda, s, C)$.
- For each $i \in [n]$, sample a fresh secret-key encryption key, i.e., $\mathsf{sk}_i \leftarrow\!\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$.
- For each $i \in [n]$, we compute a special encryption of sh'_i as follows:
  - Compute *sub-shares* of of sh'_i using SS for the same access structure, i.e., $,(\mathsf{sh}'_{i,1}, \ldots, \mathsf{sh}'_{i,n}) \leftarrow\!\!\$\ \mathsf{SS.Share}(1^\lambda, \mathsf{sh}'_i, C)$.
  - Encrypt the jth sub-share using the jth key, i.e., $\mathsf{ct}_{i,j} \leftarrow\!\!\$\ \mathsf{SKE.Enc}(\mathsf{sk}_j, \mathsf{sh}'_{i,j})$.
  - Set $\mathsf{ct}_i \leftarrow (\mathsf{ct}_{i,1}, \ldots, \mathsf{ct}_{i,n})$. This is an encryption of sh'_i with respect to the set $S = \{1, \ldots, n\}$.
- For each $i \in [n]$, assign the ith encryption key along with the special encryption of sh'_i as the share of this party. Formally, $\mathsf{sh}_i := (\mathsf{sk}_i, \mathsf{ct}_i)$.

For reconstruction, an authorized subset of parties $S \subseteq [n]$ can decrypt an authorized set of sub-shares of sh'_i for all $i \in S$, which can be used to reconstruct sh'_i. The secret can then be reconstructed using the sh'_i values for all $i \in S$.

Let us now see why our special encryption scheme indeed achieves all three properties. First, observe that to decrypt any single share, we need to decrypt an authorized subset of its sub-shares, which requires knowing the secret keys sk_i of an authorized subset of parties. Hence, an unauthorized subset of parties cannot decrypt the ciphertext of even a single share, as desired.

For the remaining properties, we first show how we can modify our encryption scheme to encrypt to arbitrary sets. To encrypt a share sh'_i with respect to a set $S \subseteq [n]$, we encrypt the jth sub-share using the jth key as above, for all parties in S. Formally, for all $j \in S$, $\mathsf{ct}_{i,j} \leftarrow\!\!\$\ \mathsf{SKE.Enc}(\mathsf{sk}_j, \mathsf{sh}'_{i,j})$. But, for all other parties, i.e., $j \in [n] \setminus S$, we encrypt some garbage value (say, 0) instead of the real sub-share $\mathsf{sh}'_{i,j}$, i.e., $\mathsf{ct}_{i,j}$ would be $\mathsf{SKE.Enc}(\mathsf{sk}_j, 0)$ for all $j \in [n] \setminus S$. This means that, the sub-shares for all parties $\in [n] \setminus S$ are essentially useless for reconstruction, hence, a share sh'_i encrypted in this manner can only be decrypted by an authorized subset of S, as desired. Lastly, to see why the third property holds, observe that an adversary that does not have the key for a party ℓ, cannot learn whether the ciphertext $c_{i,\ell}$ encrypts a real sub-share or zero, by semantic security. Hence, this adversary cannot learn whether party ℓ has been disabled or not. Our special encryption scheme can be seen as a generalization of the trivial PLBE scheme [10].

For tracing, we can simply use the tracing via jumps approach described in the previous section. Recall that the tracer is given the real shares of all n parties (since we cannot achieve tracer secrecy anyway). Hence, the tracer can use the keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and shares $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n)$ to prepare shares of all $i \in I$, wherein it encrypts sh'_i with respect to sets $\{j, \ldots, n\} \cup I$ for different values of j. By similar arguments as in the last section, the second and third properties of our special encryption imply that the tracer must find a jump at some index j^*, and this index cannot be honest. Hence, the tracer simply blames party j^*[5].

Handling Overlapping Sets. Recall that in the definition of strong traceability, the label I may contain corrupt parties. But then, the above tracing algorithm will completely break down – the tracer samples fresh ciphertexts for parties in I whenever it runs the box. But the box will immediately observe that the input shares do not match real shares of parties in $I \cap J$ and simply abort.

To solve this problem, we will exploit the fact that the tracer has access to the real shares of all n parties. At a high level, the tracer will run the reconstruction box with the real shares of I, and then replace the share of each party in I with a freshly sampled share (encrypted to set $\{1, \ldots, n\}$ like a real share), one-by-one. We consider two cases:

[5]Concurrent and independent of our work, Goyal and Waters [33] also proposed a TSS scheme for the threshold access structure, based on similar ideas, that achieves our notion of strong traceability.

1. There is a jump in behavior of the box when switching the share of a party $j^* \in \mathcal{I}$ from real to freshly-sampled: Observe that, for any honest party, the freshly-sampled share, i.e., a fresh encryption of the share to $[n]$ is distributed identically as the real share. Hence, the box cannot distinguish between the two for an honest party. Consequently, the only scenario where the box can distinguish between real and freshly-sampled shares is if it already has the real share for party j^*. This implies that j^* must be corrupt. Hence, the tracer simply blames this party.

2. There is no jump in the box's behavior: This means that the box outputs information about the correct secret with non-negligible probability, even when given freshly sampled shares of all parties in I as input. We can now rely on the same tracing procedure described above – we prepare shares of I to disable parties one-by-one and blame the party where we find a jump.

Reducing Share Size. The share of each party contains a symmetric decryption key sk_i, along with an encryption of sh'_i. If the share size for the scheme SS is w, then the encryption of sh'_i is of size $(w)^2 \cdot n$. To avoid this quadratic overhead in w, we use a hybrid encryption approach. Specifically, to encrypt sh'_i with respect to a set S, we first sample a fresh symmetric encryption key k and encrypt sh'_i with this key to get $\hat{c}_i$. Next, we encrypt the key k (which is of size $\Theta(1)$ as opposed to w) with respect to the set S, exactly as described above. This allows us to reduce share size to $w + w \cdot n$. We describe the full compiler in Section 6.

2.4 Organization

Section 3 reviews the necessary preliminaries. In Section 4, we introduce our new TSS definitions. Section 5 presents our TSS scheme achieving $\emptyset$-strong traceability and security against malicious tracers. The impossibility result is given in Appendix A. Finally, Section 6 describes our generic compiler for constructing TSS with strong traceability.

3 Preliminaries

3.1 Indistinguishability Obfuscation

An indistinguishability obfuscator Obf is a PPT algorithm that takes as input the security parameter λ and a circuit C, and outputs an obfuscated circuit C'. It must satisfy the following conditions:

Functionality. The obfuscated circuit has the same functionality as the input circuit.

Definition 3.1. *An obfuscator* Obf *satisfies Functionality if, for all* $\lambda \in \mathbb{N}$*, for all circuits* C*, and for all inputs* x*,*

$$\Pr[C'(x) = C(x) : C' \leftarrow\!\$\ \mathsf{Obf}(1^\lambda, C)] = 1.$$

Indistinguishability. For any two circuits with the same functionality and same size, their obfuscated circuits are indistinguishable.

Definition 3.2. *An obfuscator* Obf *satisfies Indistinguishability if, for all* $\lambda \in \mathbb{N}$*, all circuit ensembles* $C_0 = \{C_\lambda^{(0)}\}_\lambda$ *and* $C_1 = \{C_\lambda^{(1)}\}_\lambda$ *such that,* $\forall x, \lambda, C_\lambda^{(0)}(x) = C_\lambda^{(1)}(x)$*, and for all* PPT *adversaries* D*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathcal{A},\mathsf{Obf}}(\lambda) := |\Pr[D(\mathsf{Obf}(1^\lambda, C_\lambda^{(0)})) = 1] - \Pr[D(\mathsf{Obf}(1^\lambda, C_\lambda^{(1)})) = 1]|.$$

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

3.2 Puncturable Pseudorandom Functions

The notion of puncturable pseudorandom function (pPRF) was proposed by Sahai and Waters [51]. This is a useful tool to prove the security of cryptographic schemes based on iO.

A puncturable pseudorandom function (pPRF) PRF consists of a tuple of PPT algorithms $\mathsf{PRF} = (\mathsf{F}, \mathsf{Punc})$ satisfying the following two conditions, along with the usual notion of pseudorandomness.

Functionality preserving under puncturing. The punctured key gives the correct PRF evaluation at all the non-punctured points.

Definition 3.3. *For all $\lambda \in \mathbb{N}$, for all subsets $S \subseteq \mathsf{D}_\lambda$ of size $|S| = \mathsf{poly}(\lambda)$, and for all $x \in \mathsf{D}_\lambda \setminus S$,*

$$\Pr[\mathsf{F}(k, x) = \mathsf{F}(k_S, x) : k \leftarrow\!\!\$\ \mathcal{K}_\lambda; k_S \leftarrow \mathsf{Punc}(k, S)] = 1,$$

where $\mathsf{D} = \{\mathsf{D}_\lambda\}$ is the domain of F.

Pseudorandom at punctured points. The PRF evaluation at the punctured points is indistinguishable from random, even when the adversary is given the punctured key.

Definition 3.4. *For all $\lambda \in \mathbb{N}$, for all subsets $S = \{x_1, \ldots, x_{s(\lambda)}\} \subseteq \mathsf{D}_\lambda$ where $s(\lambda)$ is a polynomial, and for all PPT distinguishers D, the following advantage is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{sec}}_{D,\mathsf{PRF}}(\lambda) := |\Pr[D(k_S, \{\mathsf{F}(k, x_i)\}_{i\in[s(\lambda)]}) = 0] - \Pr[D(k_S, \{y_i\}_{i\in[s(\lambda)]}) = 0]|,$$

where $k \leftarrow\!\!\$\ \mathcal{K}_\lambda$, $k_S \leftarrow \mathsf{Punc}(k, S)$ and $y_i \leftarrow\!\!\$\ \mathsf{R}_\lambda$ where $\mathsf{R} = \{\mathsf{R}_\lambda\}$ is the range of F.

3.3 Access structure

Definition 3.5 (Access structure)**.** *For the set $S = \{1, \ldots, n\}$, let 2^S denote the power set of S, i.e., the set of all subsets of S. An access structure on S is a set $\mathbb{A} \subseteq 2^S \setminus \emptyset$ of non-empty subsets of S. We refer to the elements of $\mathbb{A}$ as the authorized sets and those not in $\mathbb{A}$ as the unauthorized sets. We say an access structure is* monotone *if for all sets $S_1, S_2 \in 2^S$, if $S_1 \in \mathbb{A}$ and $S_1 \subseteq S_2$, then $S_2 \in \mathbb{A}$. We say a circuit C implements an access structure $\mathbb{A}$ if C takes as input a string $x \in \{0,1\}^n$ and $C(x) = 1$ if and only if $\{i : x_i = 1\}_{i\in[n]} \in \mathbb{A}$.*

Notation. We use $[n]$ to denote the set $\{1, \ldots, n\}$. For any set $J \subseteq [n]$, we use $x_J \in \{0,1\}^n$ to denote a bit string of length n, such that for all $i \in J$, $x_i = 1$, and for all $i \in [n] \setminus J$, $x_i = 0$.

4 New Definitions of TSS

In this section, we formally define traceable secret sharing for access structures captured by monotone circuits.

Syntax. A traceable secret sharing for a circuit ensemble $\{\mathcal{C}_\lambda\}_\lambda$ is a tuple of four PPT algorithms, $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$.

- $\mathsf{Share}(1^\lambda, s, C) \to (\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk})$ is the randomized secret sharing algorithm. It takes as input the security parameter 1^λ, the access structure in the form of a poly-sized monotone circuit $C : \{0,1\}^n \to \{0,1\}$ such that $C \in \mathcal{C}_\lambda$ and the secret $s \in \mathcal{S}_\lambda$, where $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda\in\mathbb{N}}$ is the space of secrets. It outputs the secret shares of all parties $\{\mathsf{sh}_i\}$, public information pp, the tracing key tk and the verification key vk.
- $\mathsf{Rec}(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J}) \to s$ is the deterministic secret reconstruction algorithm. It takes as input secret shares of a set $J \subseteq [n]$ and outputs a secret s or $\perp$.

- $\mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\delta}, s_0, s_1, \mathcal{I}) \to (\mathcal{J}, \pi)$ is the randomized tracing algorithm. It takes as input a tracing key tk, an error parameter δ and two secrets s_0 and s_1 in $\mathcal{S}_\lambda$. It also gets oracle (black-box) access to a reconstruction box R along with a 'label' $\mathcal{I}$. It outputs a subset $\mathcal{J} \subseteq [n]$ of identities of leaking parties and an associated proof π.
- $\mathsf{Verify}(\mathsf{vk}, \mathcal{J}, \pi) \to \{0,1\}$ is the deterministic verification algorithm. It takes as input the verification key vk, an alleged traitor subset $\mathcal{J}$, and an associated proof π, and outputs a bit denoting whether the proof is valid.

Correctness. A traceable secret-sharing scheme for general access structures must satisfy correctness, which states that the shares of any authorized set of parties J i.e., $C(x_J) = 1$ should suffice to reconstruct the secret.

Definition 4.1. *Let* $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme. We say that* GTSS *is* ***correct*** *if for every* $\lambda \in \mathbb{N}$*, every secret* $s \in \mathcal{S}_\lambda$*, every* $n \in \mathbb{N}$*, every monotone circuit* $C \in \mathcal{C}_\lambda$*, and every subset* $J = \{i_1, \ldots, i_t\} \subseteq [n]$ *such that* $C(x_J) = 1$*, it holds that*

$$\Pr\left[s' = s\right] \geq 1 - \mathsf{negl}(\lambda),$$

where $(\{\mathsf{sh}_1, \ldots, \mathsf{sh}_n\}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Share}(1^\lambda, s, C)$ *and* $s' := \mathsf{Rec}(\mathsf{pp}, \mathsf{sh}_{i_1}, \ldots, \mathsf{sh}_{i_t})$.

Security. A traceable secret-sharing scheme should satisfy four security properties: secrecy, tracer secrecy, $\emptyset$-strong traceability, and tracer non-imputability. The first is the standard secrecy property of secret-sharing schemes, which stipulates that any subset of parties that does not satisfy C should learn nothing about the secret s. This is captured by the following definition. Note that we focus on the computational version, wherein we require secrecy to hold against all PPT adversaries.

Definition 4.2 (Secrecy)**.** *Let* $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme for secrets in* $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda \in \mathbb{N}}$*. We say that* GTSS *satisfies* ***secrecy*** *if for every* $\lambda \in \mathbb{N}$*, any two secrets* $s_0, s_1 \in \mathcal{S}_\lambda$*, every monotone circuit* $C : \{0,1\}^n \to \{0,1\}$ *such that* $C \in \mathcal{C}_\lambda$*, and every subset* $\mathcal{J} \subseteq [n]$ *such that* $C(x_\mathcal{J}) = 0$*, every* PPT *algorithm* D*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{sec}}_{D,\mathsf{GTSS}}(\lambda) := \left| \Pr\left[b^* = b : \begin{array}{c} b \leftarrow\!\!\$\ \{0,1\} \\ (\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Share}(1^\lambda, s_b, C) \\ b^* \leftarrow\!\!\$\ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}) \end{array} \right] - \frac{1}{2} \right|$$

Tracer Secrecy. Tracer secrecy requires that secrecy is maintained even against the tracer. That is, that the tracing key tk does not help in breaking secrecy. This is made formal in the following definition.

Definition 4.3 (Tracer Secrecy)**.** *Let* $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme for secrets in* $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda \in \mathbb{N}}$*. We say that* GTSS *satisfies* ***tracer secrecy*** *if for every* $\lambda \in \mathbb{N}$*, any two secrets* $s_0, s_1 \in \mathcal{S}_\lambda$*, every monotone circuit* $C : \{0,1\}^n \to \{0,1\}$ *such that* $C \in \mathcal{C}_\lambda$*, and every subset* $\mathcal{J} \subseteq [n]$ *such that* $C(x_\mathcal{J}) = 0$*, every* PPT *algorithm* D*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{GTSS}}(\lambda) := \left| \Pr\left[b^* = b : \begin{array}{c} b \leftarrow\!\!\$\ \{0,1\} \\ (\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Share}(1^\lambda, s_b, C) \\ b^* \leftarrow\!\!\$\ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{tk}, \mathsf{vk}) \end{array} \right] - \frac{1}{2} \right|$$

$\emptyset$-Strong Traceability. Suppose an unauthorized set of parties $\mathcal{J} \subseteq [n]$ of parties i.e., $C(x_\mathcal{J}) = 0$, constructs a reconstruction box R using their shares. This R is an algorithm that takes in secret shares of all parties in a set $\mathcal{I}$ (we call this the label of the box) and can distinguish whether the shares correspond to secret s_0 or s_1. Intuitively, if this R is a "minimally-useful" reconstruction box, then it should be possible

Experiment $\mathbf{Exp\emptyset StrongTrace}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda)$

$(C, \mathcal{J}, s_0, s_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$ // $\mathcal{A}$ chooses the parties to corrupt
$b \leftarrow\!\$ \{0,1\}$
$(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\$ \mathsf{Share}(1^\lambda, s_b, C)$
$(R, \mathcal{I}) \leftarrow\!\$ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}})$
$(\mathcal{J}^*, \pi) \leftarrow\!\$ \mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I})$
$\mathsf{GoodBox} := (\mathcal{I} \cap \mathcal{J} = \emptyset) \wedge C(x_\mathcal{J}) = 0 \wedge C(x_\mathcal{I}) = 0 \wedge$
$\qquad (R \text{ is } (C, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{pp}, , s_0, s_1, b, \epsilon)\text{-minimally-useful})$
if $\mathcal{J}^* \not\subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 1
if $\neg\mathsf{GoodBox}$ **then return** 0
if $\mathcal{J}^* \neq \emptyset \wedge \mathcal{J}^* \subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 0 **else return** 1

Figure 3: The $\emptyset$-strong tracing experiment for a traceable secret-sharing scheme GTSS and an adversary $\mathcal{A}$.

to trace it back to at least one of the parties who "contributed" its share to it. We require that tracing should be done given only black-box access to R (and the label).

In more detail, say that the secret shared among the parties is s_b. Then, R is a minimally-useful reconstruction box with respect to a label $\mathcal{I}$ if it guesses b with non-negligible probability, when given shares of s_b for the set $\mathcal{I}$ as input. Definition 4.4 below formally defines such reconstruction boxes.

Definition 4.4 (Minimally-useful Reconstruction Box)**.** *Let* GTSS *be a traceable secret-sharing scheme. Let* $\lambda \in \mathbb{N}$*, let* $C : \{0,1\}^n \to \{0,1\}$ *be a monotone circuit. For* $\epsilon \in [0,1]$*, secrets* $s_0, s_1 \in \mathcal{S}_\lambda$*, a bit* $b \in \{0,1\}$*, a set* $\mathcal{J} \subseteq [n]$ *such that* $C(x_\mathcal{J}) = 0$*, and corresponding shares* $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$*, and a label* $\mathcal{I} = \{i_1, \ldots, i_f\}$*, a reconstruction box* R *is* $(C, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{pp}, s_0, s_1, b, \epsilon)$*-minimally-useful if,*

$$\Pr\left[R(\mathsf{sh}_{i_1}, \ldots, \mathsf{sh}_{i_f}) = b\right] \geq \frac{1}{2} + \epsilon,$$

where $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\$ \mathsf{Share}(1^\lambda, s_b, C)$ *and the probability is taken over the random coins of* R*.*

Equipped with this notion, the $\emptyset$-Strong traceability experiment is presented in Fig. 3 and the definition is given in Definition 4.5. The game starts with the adversary sending the access structure C, the set of corrupt parties $\mathcal{J}$ and two secret s_0, s_1. The challenger then samples $b \leftarrow\!\$ \{0,1\}$, and generates secret shares of s_b. It sends shares of $\mathcal{J}$ along with the public information pp to the adversary, which then responds with a reconstruction box R and a label $\mathcal{I}$. We require that if this box is indeed minimally-useful, then the Trace algorithm should find at least one corrupt party with a valid proof, and never blame an honest party.

Definition 4.5 ($\emptyset$-Strong Traceability)**.** *Let* $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme, and let* $\epsilon = \epsilon(\lambda)$ *be a function of the security parameter. We say that* GTSS *is* ***traceable****, if for every probabilistic polynomial time adversary* $\mathcal{A}$*, the following function is negligible in* λ*:*

$$\mathsf{Adv}^{\emptyset-\mathrm{str}-\mathrm{trac}}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) := \Pr\left[\mathbf{Exp\emptyset StrongTrace}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) = 1\right].$$

Remark 4.1 (On the $\emptyset$-Strong traceability definition)**.** *Some remarks are in order:*

- **Requiring** $C(x_\mathcal{J}) = 0$**:** *Note that we require the tracer to find at least one traitor only if* $C(x_\mathcal{J}) = 0$ *– this is essentially optimal because if the adversary corrupts an authorized set* $\mathcal{J}$*, i.e., if* $C(x_\mathcal{J}) = 1$*, then the adversary can just reconstruct the secret and put that in the box – such a box is untraceable.*

Experiment $\mathbf{ExpNI}_{\mathcal{A},\mathsf{GTSS}}$

1 : $(C, i^*, s, \mathsf{state}) \leftarrow\!\$\ \mathcal{A}(1^\lambda)$

2 : $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\$\ \mathsf{Share}(1^\lambda, s, C)$

3 : $(\mathcal{J}^*, \pi) \leftarrow\!\$\ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in[n]\setminus\{i^*\}}, \mathsf{tk}, \mathsf{vk})$

4 : **if** $i^* \in \mathcal{J}^* \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 1 **else return** 0

Figure 4: The tracer non-imputability experiment for GTSS and an adversary $\mathcal{A}$.

- **Requiring $C(x_{\mathcal{I}}) = 0$:** *This is a natural requirement because if the input shares to the box already satisfy the access structure, then the box is not useful.*
- **Requiring $\mathcal{I} \cap \mathcal{J} = \emptyset$.** *We require tracing to work only if the label does not contain corrupt parties. But we do not require $\mathcal{I} \cup \mathcal{J}$ to be minimal. As explained in Section 1, this is stronger than the definitions in previous works. In Section 4.1, we give an even stronger definition of tracing where we lift this restriction.*
- **Private tracing:** *In the tracing experiment, the adversary is not given access to the tracing key* tk *and the verification key* vk. *This is because, for our constructions, knowledge of these keys can allow an adversary to evade tracing. We leave the problem of constructing secret sharing schemes with public tracing as an avenue for future work.*

Tracer Non-imputability. Lastly, we require that a traceable secret-sharing scheme satisfy the notion of tracer non-imputability, stating that honest parties cannot be wrongly blamed by a cheating tracer. In more detail, this means that even a malicious tracer cannot produce an accepting proof for the culpability of an honest party. This is formally captured by the security experiment in Fig. 4. There, the adversary gets the tracing and verification keys tk and vk, and may obtain the secret shares of any party of its choosing, as long as it is not the honest party it tries to blame.

Definition 4.6. *Let* $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a TSS scheme. We say that* GTSS *achieves* **tracer non-imputability**, *if for every probabilistic polynomial time adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{ni}}_{\mathcal{A},\mathsf{GTSS}}(\lambda) := \Pr\left[\mathbf{ExpNI}_{\mathcal{A},\mathsf{GTSS}}(\lambda) = 1\right],$$

where the experiment $\mathbf{ExpNI}_{\mathcal{A},\mathsf{GTSS}}$ *is as defined in Figure 4.*

4.1 Strong traceability for TSS

We now define a stronger notion of traceability, which says that tracing should work even if the box can lie about the label $\mathcal{I}$. Compared to the definition of $\emptyset$-Strong traceability (as given in Definition 4.5), here we allow the adversary to include corrupt parties in the label $\mathcal{I}$ i.e., $\mathcal{I} \cap \mathcal{J}$ may not be empty. We formalize this notion in Definition 4.7.

Definition 4.7 (Strong Traceability)**.** *Let* $\mathsf{GTSS} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme. We say that* GTSS *is* ***strong-traceable****, if for every probabilistic polynomial time adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{str-trac}}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) := \Pr\left[\mathbf{ExpStrongTrace}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) = 1\right],$$

where the experiment $\mathbf{ExpStrongTrace}_{\mathcal{A},\mathsf{GTSS},\epsilon}$ *is as defined in Figure 5.*

Experiment $\mathbf{ExpStrongTrace}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda)$

$(C, \mathcal{J}, s_0, s_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$ // $\mathcal{A}$ chooses the parties to corrupt
$b \leftarrow\$ \{0,1\}$
$(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\$ \mathsf{Share}(1^\lambda, s_b, C)$
$(R, \mathcal{I}) \leftarrow\$ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}})$
$(\mathcal{J}^*, \pi) \leftarrow\$ \mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I})$
$\mathsf{GoodBox} := C(x_\mathcal{J}) = 0 \wedge C(x_\mathcal{I}) = 0 \wedge$
$\qquad (R \text{ is } (C, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, s_0, s_1, b, \epsilon)\text{-minimally-useful})$
if $\mathcal{J}^* \not\subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 1
if $\neg\mathsf{GoodBox}$ **then return** 0
if $\mathcal{J}^* \neq \emptyset \wedge \mathcal{J}^* \subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 0 **else return** 1

Figure 5: The strong tracing experiment for a traceable secret-sharing scheme GTSS and an adversary $\mathcal{A}$. The key difference from the $\emptyset$-strong tracing experiment is that the label of the box I may overlap with the corrupt parties, i.e., $I \cap J$ is not required to be $\emptyset$.

5 GTSS with $\emptyset$-Strong Traceability from iO

We now present our traceable secret sharing scheme for monotone circuits which achieves $\emptyset$-strong traceability, tracer secrecy and tracer non-imputability.

5.1 Building Block: Doubly-Constrained Signatures

In this section, we define a new primitive that we call doubly-constrained signatures. Note that while this primitive is related to constrained signatures [13] and splittable signatures [42], the notion we define and achieve has not been considered before.

Informally, a doubly constrained signature scheme allows the computation, given circuits C_1 and C_2, of (i) a constrained public key pk_{C_1} and a corresponding signing key sk_{C_1} and (ii) a constrained signing key sk_{C_2}. The public key pk_{C_1} has the property that for any message x such that $C_1(x) = 0$, there are no valid signatures on x relative to pk_{C_1}. For security, we require that pk_{C_1} is indistinguishable from a valid, honestly generated public key for any adversary that can query signatures on any x such that $C_1(x) = 1$. The constrained signing key sk_{C_2} can only be used to sign messages x such that $C_2(x) = 1$. Moreover, signatures generated using sk_{C_2} should be indistinguishable from those generated using sk_{C_1}. Note that we do allow the public key and secret key to be constrained by different circuits, i.e., C_1 may not be equal to C_2.

More formally, a doubly-constrained signature scheme for a circuit ensemble $\{\mathcal{C}_\lambda\}_\lambda$ is a tuple of four PPT algorithms $\mathsf{DCS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SignVerify}, \mathsf{ConstPK})$ where:

- $\mathsf{KeyGen}(1^\lambda, \ell, z, C_1) \to (\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_{C_1})$ takes as input the security parameter λ, an upper bound on the message length ℓ, a circuit $C_1 : \{0,1\}^\ell \to \{0,1\}$ in $\mathcal{C}_\lambda$ and an upper bound $z = \max_{C\in\mathcal{C}_\lambda} |C|$ on the size of the circuits handled by the scheme. It outputs a valid (unconstrained) secret/public key pair $(\mathsf{pk}, \mathsf{sk})$ along with a constrained secret key sk_{C_1}. We will use $\mathsf{len}_\sigma = \mathsf{len}_\sigma(\lambda, \ell, z)$ to denote the length of a signature for the scheme.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \{0,1\}^{\mathsf{len}_\sigma}$ is the randomized algorithm that takes as input the secret signing key and a message m, and outputs a signature σ_m.

- $\mathsf{SignVerify}(\mathsf{pk}, m, \sigma_m) \to \{0,1\}$ is the deterministic algorithm that takes as input the public key, the message and a signature, and outputs whether the signature is valid.
- $\mathsf{ConstPK}(1^\lambda, \ell, z, C_1, C_2) \to (\mathsf{pk}_{C_2}, \mathsf{sk}_{C_2}, \mathsf{sk}_{C_1})$ takes as input the security parameter, message length ℓ, an upper bound z on the size of the circuits, and two circuits $C_1, C_2 \in \mathcal{C}_\lambda$. It outputs a constrained public key-secret key pair $(\mathsf{pk}_{C_2}, \mathsf{sk}_{C_2})$ along with a constrained secret key sk_{C_1}.

Correctness. We require two notions of correctness. First, an honestly generated signature should verify successfully with the corresponding public key. This should hold for the key pairs generated by KeyGen as well as ConstPK. Second, we require that for any m that does not satisfy the circuit C_2, there should be no valid signature with respect to pk_{C_2}. Definitions 5.1 and 5.2 formalize these notions.

Definition 5.1. *A Doubly constrained signature scheme* DCS *is correct if, for all λ, for all $\ell, z \in \mathbb{N}$, for all circuits C_1 of size bounded by z, we have,*

$$\Pr[\mathsf{SignVerify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1 \; \forall \; m \in \{0,1\}^\ell$$
$$\Pr[\mathsf{SignVerify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}_{C_1}, m)) = 1] = 1 \; \forall \; m \in \{0,1\}^\ell \; \textit{s.t.} \; C_1(m) = 1$$

where, $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_{C_1}) \leftarrow\!\!\$ \; \mathsf{KeyGen}(1^\lambda, \ell, z, C_1)$ and the probability is over the randomness of the Sign *algorithm.*

Definition 5.2. *A Doubly constrained signature scheme* DCS *is constrained-correct if, for all λ, for all $\ell, z \in \mathbb{N}$, for all circuits C_1, C_2 of size bounded by z, we have,*

$$\Pr[\mathsf{SignVerify}(\mathsf{pk}_{C_2}, m, \mathsf{Sign}(\mathsf{sk}_{C_2}, m)) = 1] = 1 \; \forall \; m \in \{0,1\}^\ell \; \textit{s.t.} \; C_2(m) = 1$$
$$\Pr[\mathsf{SignVerify}(\mathsf{pk}_{C_2}, m, \mathsf{Sign}(\mathsf{sk}_{C_1}, m)) = 1] = 1 \; \forall \; m \in \{0,1\}^\ell \; \textit{s.t.} \; C_1(m) = C_2(m) = 1$$
$$\mathsf{SignVerify}(\mathsf{pk}_{C_2}, m, \sigma) = 0 \; \forall \; m \in \{0,1\}^\ell \; \textit{s.t.} \; C_2(m) = 0 \wedge \forall \; \sigma \in \{0,1\}^{\mathsf{len}_\sigma},$$

where $(\mathsf{pk}_{C_2}, \mathsf{sk}_{C_2}, \mathsf{sk}_{C_1}) \leftarrow\!\!\$ \; \mathsf{ConstPK}(1^\lambda, \ell, z, C_1)$ and the probability is over the randomness of the Sign *algorithm.*

Security. We require two key properties. First, the distribution of signatures generated by a constrained signing key sk_{C_1} should be indistinguishable from that generated by an unconstrained key sk, for all messages such that $C_1(m) = 1$. Definition 5.3 formalizes this notion.

Definition 5.3. *A doubly-constrained signature scheme* DCS *is said to have identically-distributed signatures if the following advantage is negligible for all* PPT *adversaries:*

$$\mathsf{Adv}^{\mathrm{id-dist}}_{\mathcal{A},\mathsf{DCS}}(\lambda) := \left| \Pr[\mathbf{ExpIdDist}_{\mathcal{A},\mathsf{DCS}}(\lambda) = 1] - \frac{1}{2} \right|,$$

where the experiment $\mathbf{ExpIdDist}_{\mathcal{A},\mathsf{DCS}}$ is as defined in Figure 6.

The second property requires that an adversary cannot distinguish between the constrained public key pk_{C_2} and a normal public key pk, even if the adversary is allowed to query signatures on arbitrary messages m that satisfy $C_2(m) = 1$. Moreover, if C_1 is more restrictive than C_2, then the adversary additionally gets access to the constrained signing key sk_{C_1}, but it should still not be able to distinguish between the two public keys. Formally, a circuit C_1 is more restrictive than C_2 if $C_1(m) = 1 \implies C_2(m) = 1$. In other words, the set of messages accepted by C_1 is a subset of the messages accepted by C_2, i.e., $\{m : m \in \{0,1\}^\ell \wedge C_1(m) = 1\} \subseteq \{m : m \in \{0,1\}^\ell \wedge C_2(m) = 1\}$. We will use $C_1 \implies C_2$ as a short-hand for this. Figure 7 and Definition 5.4 formalize a game-based definition for this notion. Note that this is a strict generalization of the security definition for constrained signatures as defined in [13].

Experiment $\mathbf{ExpIdDist}_{\mathcal{A},\mathsf{DCS}}$	Oracle $\mathsf{SignO}(m)$
1 : $(\ell, z, C_1, \mathsf{state}) \leftarrow\$ \mathcal{A}(1^\lambda)$	1 : **if** $C_1(m) = 0$ **then**
2 : $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_{C_1}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda, \ell, z, C_1)$	2 : **return** $\mathsf{Sign}(\mathsf{sk}, m)$
3 : $b \leftarrow\$ \{0,1\}$; $\mathsf{sk}^* := \mathsf{sk}$ **if** $b = 0$ **else** sk_{C_1}	3 : **else return** $\mathsf{Sign}(\mathsf{sk}^*, m)$
4 : $b^* \leftarrow\$ \mathcal{A}^{\mathsf{SignO}(\cdot)}(\mathsf{state}, \mathsf{pk})$	
5 : **return** 1 **if** $b = b^*$ **else** 0	

Figure 6: The Identical-Distribution property for normal keys, of a doubly-constrained signature scheme DCS.

Experiment $\mathbf{G}^{sec}_{\mathcal{A},\mathsf{DCS}}$	Oracle $\mathsf{SignO}(m)$
1 : $(\ell, z, C_1, C_2, \mathsf{state}) \leftarrow\$ \mathcal{A}(1^\lambda)$	1 : **if** $C_2(m) = 0$ **then**
2 : $b \leftarrow\$ \{0,1\}$	2 : **return** $\perp$
3 : **if** $b = 0$ **then**	3 : **return** $\mathsf{Sign}(\mathsf{sk}, m)$
4 : $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_{C_1}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda, \ell, z, C_1)$	
5 : **else**	
6 : $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_{C_1}) \leftarrow\$ \mathsf{ConstPK}(1^\lambda, \ell, z, C_1, C_2)$	
7 : $b^* \leftarrow\$ \mathcal{A}^{\mathsf{SignO}(\cdot)}(\mathsf{state}, \mathsf{pk}, \boxed{\mathsf{sk}_{C_1} \text{ if } C_1 \implies C_2})$	
8 : **return** 1 **if** $b = b^*$ **else** 0	

Figure 7: The security game for a doubly-constrained signature scheme DCS.

Definition 5.4. *A doubly constrained signature* DCS *is secure if for all* $\lambda \in \mathbb{N}$*, and all* PPT *adversaries* $\mathcal{A}$*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{A},\mathsf{DCS}}(\lambda) := \left| \Pr[\mathbf{G}^{sec}_{\mathcal{A},\mathsf{DCS}}(\lambda) = 1] - \frac{1}{2} \right|$$

where the game $\mathbf{G}^{sec}_{\mathcal{A},\mathsf{DCS}}$ *is as defined in Figure 7.*

5.2 Our Construction

We start with describing a simpler construction that does not achieve tracer non-imputability. In Section 5.3, we show how the public parameters for this scheme can be made re-usable across multiple secrets. Lastly, in Section 5.4, we describe how to modify this scheme to achieve tracer non-imputability.

We use a puncturable $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\} \times \{0,1,\ldots,n\}$, a doubly constrained signature scheme DCS and indistinguishability obfuscation Obf as building blocks. Figure 8 gives a formal description. We do not describe a Verify algorithm since this scheme does not provide tracer non-imputability.

Correctness. Correctness follows directly from correctness of the doubly-constrained signature and obfuscation. We formally prove this below.

Proof of Correctness for iOGTSS. For any set $J \subseteq [n]$ such that $C(x_J) = 1$, when the obfuscated program $\hat{\mathsf{P}}$ is run with $\{\mathsf{sh}_i = (i, \mathsf{ct}_i, \sigma_i)\}_{i \in J}$ as input, all the signatures σ_i will verify successfully by correctness of the DCS. Moreover, since each ciphertext ct_i decrypts to $(0,0,0)$, we will get $S = [n]$. Hence, $\hat{J} = J$ and

$\underline{\mathsf{Share}(1^\lambda, s, C)}$:

1. Sample PRF key $k \leftarrow\!\!\$ \{0,1\}^\lambda$ and doubly-constrained signature keys $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_1) \leftarrow\!\!\$ \mathsf{DCS.KeyGen}(1^\lambda, \ell, z, C_1)$ where $C_1 : [n] \times \mathsf{CT} \to \{0,1\}$ is defined as, $C_1(i, \mathsf{ct}) = 1$ if and only if, $\mathsf{Dec}(k, \mathsf{ct}) = (1, \cdot, \cdot)$. / $\mathsf{CT} := \{0,1\}^\lambda \times \{0,1\}^{1+\log(n+1)+\log(|\mathcal{S}_\lambda|)}$ and $\ell = \ell(n, \lambda)$, $z = z(\lambda, n)$ will be specified later.
2. For all $i \in [n]$, compute $\mathsf{ct}_i \leftarrow\!\!\$ \mathsf{Enc}(k, (0,0,0))$ and $\sigma_i := \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}_i))$. Set $\mathsf{sh}_i \leftarrow (i, \mathsf{ct}_i, \sigma_i)$.
3. Obfuscate the program P described below, i.e., $\hat{\mathsf{P}} \leftarrow \mathsf{Obf}(\mathsf{P})$.
4. Output $\{\mathsf{sh}_i\}_{i\in[n]}$, $\mathsf{pp} \leftarrow \hat{\mathsf{P}}$, $\mathsf{tk} \leftarrow (k, \mathsf{sk}_1)$, $\mathsf{vk} \leftarrow \bot$.

$\underline{\mathsf{Enc}(k, (f, j, m))}$:

1. Sample $r \leftarrow\!\!\$ \{0,1\}^\lambda$. Output $\mathsf{ct} \leftarrow (r, \mathsf{ct}' \leftarrow \mathsf{PRF.F}(k, r) + (f||j||m)) \in \mathsf{CT}$.

$\underline{\mathsf{Dec}(k, \mathsf{ct})}$:

1. Parse ct as (r, ct') and output $\mathsf{ct}' - \mathsf{PRF.F}(k, r)$.

$\underline{\mathsf{P}(\{i, \mathsf{ct}_i, \sigma_i\}_{i\in J})}$:

1. **Constants**: C, k, pk, s.
2. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
3. For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$. Compute $S_i := \{j_i + 1, \ldots, n\}$ and $S := \bigcap_{i\in\hat{J}} S_i$.
4. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
5. Set $\hat{J} = \hat{J} \cap S$. Abort if $C(x_{\hat{J}}) = 0$.
6. If $\forall\, i \in \hat{J}$, $f_i = 0$, then output s.
7. Otherwise, output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

$\underline{\mathsf{Rec}(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J})}$:

1. Parse pp as $\hat{\mathsf{P}}$. Output $\hat{\mathsf{P}}(\{\mathsf{sh}_i\}_{i\in J})$.

$\underline{\mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I})}$:

1. Parse tk as (k, sk_1). Sample $b^* \leftarrow\!\!\$ \{0,1\}$, and set $s^* := s_{b^*}$.
2. Let $\mathcal{J}^* := \bot$. For $j \in \{0, \ldots, n\}$:
   (a) Let $p_j \leftarrow 0$. For $q \in \{1, \ldots, N\}$: / $N = \frac{8\lambda\cdot n^2}{\epsilon^2}$
      i. For all $i \in \mathcal{I}$, compute $\mathsf{ct}'_i \leftarrow \mathsf{Enc}(k, (1, j, s^*))$ and $\sigma'_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}_1, (i, \mathsf{ct}'_i))$.
      ii. Query R on $\{\mathsf{sh}^*_{i,j} \leftarrow (i, \mathsf{ct}'_i, \sigma'_i)\}_{i\in\mathcal{I}}$. Let b denote R's response.
      iii. If $s_b = s^*$, then set $p_j \leftarrow p_j + 1$.
   (b) Compute $p'_j \leftarrow p_j/N$. If $j > 0$ and $p_{j-1} - p_j > \epsilon/4n$, then set $\mathcal{J}^* \leftarrow \mathcal{J}^* \cup \{j\}$.

Figure 8: Our iO-based scheme iOGTSS with $\emptyset$-strong traceability and tracer secrecy. In Section 5.4, we show how to modify this scheme to also achieve tracer non-imputability.

the program will not abort in line 5, and since $f_i = 0$ for all input shares, the program will indeed output the correct secret s. □

Tracer Secrecy. We directly prove tracer secrecy since it also implies secrecy. We prove tracer secrecy of our scheme based on security of the doubly-constrained signature scheme and the indistinguishability

obfuscation.

Theorem 5.5. *For all* PPT *adversaries* D*, there exist* PPT *adversaries* D_1, D_2, D_3, D_4*, such that,*

$$\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS}}(\lambda) \leq \mathsf{Adv}^{\mathrm{sec}}_{D_1,\mathsf{DCS}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_2,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_3,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_4,\mathsf{Obf}}(\lambda)$$

Proof of Theorem 5.5. We prove the theorem via a series of hybrids. Let us denote the original game by H_0.

We now define a hybrid H_1, where instead of using DCS.KeyGen to generate keys, the challenger uses DCS.ConstPK, with C_1 as defined in line 1 of the Share algorithm, and C_2 defined as follows. For any (i, ct), $C_2(i, \mathsf{ct}) = 1$ if and only if, either $i \in \mathcal{J}$ or $\mathsf{Dec}(\mathsf{ct}) = (1, \cdot, \cdot)$. In other words, for honest parties $i \in [n] \setminus \mathcal{J}$, valid signatures exist only on ciphertexts that encrypt messages with first bit equal to one.

We claim that if an adversary can distinguish between H_0 and H_1, it can be used to break security of the DCS scheme. To see why, observe that (i) C_1 is indeed more restrictive than C_2, i.e., $C_1 \implies C_2$, and (ii) the adversary only sees the constrained secret key sk_{C_1} and signatures which are included in shares of the corrupt parties $\mathcal{J}$, i.e., signatures on messages which satisfy the circuit C_2.

More formally, we will construct an adversary D_1 that breaks security of DCS from D that can distinguish between the two hybrids. D_1 acts as the challenger to D, and samples $b_1 \leftarrow\!\!\$\ \{0,1\}$, and sets $s \leftarrow s_{b_1}$.

D_1 then samples a PRF key k, and gives as input ℓ, z, C_1 and C_2 as defined above, to its challenger. It then gets $(\mathsf{pk}, \mathsf{sk}_{C_1})$ from its challenger and computes shares of all parties $i \in \mathcal{J}$. For each share, it computes the ciphertext $\mathsf{ct}_i \leftarrow\!\!\$\ \mathsf{Enc}(k, (0,0,0))$, and then queries the SignO of its challenger to get a signature on (i, ct_i). It computes tk and obfuscates the program as given in Fig. 8. D_1 then sends $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{tk}$ and pp to its adversary. It then forwards D's response to its challenger. It is clear that D_1 perfectly simulates the game to D, and hybrid H_0 corresponds to $b = 0$ in the DCS security game, and H_1 corresponds to $b = 1$. Hence, we get that,

$$|\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_0}(\lambda) - \mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_1}(\lambda)| = \mathsf{Adv}^{\mathrm{sec}}_{D_1,\mathsf{DCS}}(\lambda).$$

Next, we define a hybrid H_2, wherein, we change the program P as follows. We hard-code $\mathcal{J}$ in P, and, we abort if the input set J is a subset of $\mathcal{J}$, i.e., if $J \subseteq \mathcal{J}$. Figure 9 presents this new program. We argue that it is hard to distinguish between hybrids H_1 and H_2 because the functionality of the obfuscated program is the same in both. To see that, recall that $C(x_{\mathcal{J}}) = 0$, so, in hybrid H_1, if the program P is run with input $J \subseteq \mathcal{J}$, it would always be aborted in line 5, because $\hat{J} \subseteq J \subseteq \mathcal{J}$, which implies $C(x_{\hat{J}}) = 0$. Hence, it does not change functionality if we simply abort for such inputs.

More formally, if the advantage of D in hybrids H_1 and H_2 differ by a lot, we can use D to construct an adversary D_2 which can break iO security. We get,

$$|\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_1}(\lambda) - \mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_2}(\lambda)| \leq \mathsf{Adv}^{\mathrm{iosec}}_{D_2,\mathsf{Obf}}(\lambda).$$

We now make another hybrid H_3, wherein we change the program P_2 to abort if for all non-corrupt input shares, i.e., $J \setminus \mathcal{J}$, the ciphertext decrypts to $(0, \cdot, \cdot)$ or the signature is not valid. Figure 9 depicts this change. Again, we argue that this does not change the functionality of the obfuscated program. To see that, observe that because the public key of DCS has been constrained, for parties $i \in J \setminus \mathcal{J}$, no valid signatures exist on ciphertexts which encrypt $(0, \cdot, \cdot)$. This means, that even in H_2, the program when given an input J such that for all these parties, the ciphertext encrypts $(0, \cdot, \cdot)$, we know that none of these signatures would verify, and hence they would not be included in $\hat{J}$. This implies that for such input sets J, the resulting $\hat{J}$ would end up being $\subseteq \mathcal{J}$ – which means $C(x_{\hat{J}}) = 0$, so such inputs would anyway be aborted in the second last line. Hence, adding the abort line in the beginning does not affect the functionality. More formally, we get,

$$|\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_2}(\lambda) - \mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_3}(\lambda)| \leq \mathsf{Adv}^{\mathrm{iosec}}_{D_3,\mathsf{Obf}}(\lambda).$$

We define another hybrid H_4 where the program does not contain s. To see why this does not change the functionality, observe that in H_3, the only valid inputs to the program which would ever reach the last line are those where $J \setminus \mathcal{J} \neq \emptyset$, and, for at least one party i^* in $J \setminus \mathcal{J}$, the signature is valid and the decryption $f_i = 1$. Hence, as per the logic of the program, i^* must be in $\hat{J}$. This means, that in the last line, the program will never output s, and just output m_{i^*}. In other words, the secret s is never used. We formally depict this program in Fig. 10. More formally,

$$|\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_3}(\lambda) - \mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS},H_4}(\lambda)| \leq \mathsf{Adv}^{\mathrm{iosec}}_{D_4,\mathsf{Obf}}(\lambda).$$

Lastly, we observe that the view of the adversary is now completely independent of the secret s_{b_1}, so the adversary can guess b_1 with probability bounded by $1/2$. This proves the theorem.

$\underline{\mathsf{P}(\{\mathsf{sh}_i\}_{i\in J})}$:

1. **Constants**: C, k, pk, s and $\mathcal{J}$.
2. Abort if $J \subseteq \mathcal{J}$. / in Hybrids H_2 onward
3. Abort if for all $i \in J \setminus \mathcal{J}$, $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$ or $\mathsf{Dec}(k, \mathsf{ct}_i) = (0, \cdot, \cdot)$. / in Hybrids H_3 onwards
4. Parse ct_i as (r, ct'_i).
5. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
6. For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
7. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1, \ldots, n\}$. Compute $S \leftarrow \bigcap_{i\in\hat{J}} S_i$.
8. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
9. Set $\hat{J} = \hat{J} \cap S$.
10. Abort if $C(x_{\hat{J}}) = 0$.
11. If $\forall\, i \in \hat{J}$, $f_i = 0$, then output s.
12. Otherwise, output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

Figure 9: The modified programs P used in hybrids H_2 and H_3 in the tracer-secrecy proof of our scheme iOGTSS.

□

∅-Strong Traceability. We prove ∅-Strong traceability in Theorem 5.6 below, by relying on the security of the signature scheme DCS, the obfuscation scheme Obf, and the puncturable PRF. Note that we focus on a less-adaptive notion of ∅-Strong traceability, wherein the adversary sends the label $\mathcal{I}$ along with the corrupt set $\mathcal{J}$ in the first step of the game. We refer to this modified game as semi-adaptive tracing, and define advantage of an adversary $\mathcal{A}$ as:

$$\mathsf{Adv}^{\mathrm{sa-\emptyset-str-trac}}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) := \Pr[\mathbf{Exp\emptyset StrongTrace}^{sa}_{\mathcal{A},\mathsf{GTSS},\epsilon} = 1].$$

$\underline{\mathsf{P}(\{\mathsf{sh}_i\}_{i\in J})}$:

1. **Constants**: C, k, pk and $\mathcal{J}$.
2. Abort if $J \subseteq \mathcal{J}$.
3. Abort if for all $i \in J \setminus \mathcal{J}$, $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$ or $\mathsf{Dec}(k, \mathsf{ct}_i) = (0, \cdot, \cdot)$.
4. Parse ct_i as (r, ct'_i).
5. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
6. For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
7. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1, \ldots, n\}$. Compute $S \leftarrow \bigcap_{i\in\hat{J}} S_i$.
8. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
9. Set $\hat{J} = \hat{J} \cap S$.
10. Abort if $C(x_{\hat{J}}) = 0$.
11. Output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

Figure 10: The modified program P used in hybrid H_4 in the tracer-secrecy proof of our scheme iOGTSS.

Theorem 5.6. *If* DCS, Obf *and the puncturable* PRF *schemes are secure, then, for all* PPT *adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathrm{sa}-\emptyset-\mathrm{str}-\mathrm{trac}}_{\mathcal{A},\mathsf{iOGTSS},\epsilon}(\lambda) = \mathsf{negl}(\lambda).$$

Proof of Thm. 5.6. We prove the theorem in two parts. In the first part, we prove that the tracer will blame at least one party if the reconstruction box R is minimally-useful. In the second part, we prove that the blamed party cannot be honest.

We start with the first part. Let us use s_0, s_1 to denote the secrets given by $\mathcal{A}$. Let $s = s_b$ denote the secret that is shared in the Share algorithm, and let $s^* = s_{b^*}$ denote the second secret that is sampled by the tracer, where both b and b^* are sampled uniformly at random from $\{0, 1\}$. Let $\mathcal{I}$ denote the label of the reconstruction box R, and let $\mathcal{J}$ denote the set of corrupt parties.

We first prove that if the reconstruction box is minimally-useful, i.e., it outputs b with non-negligible probability when given honestly generated shares for the set $\mathcal{I}$, then it must output b^* with roughly the same probability, when given tracer-generated shares of the same set $\mathcal{I}$.

Lemma 5.7. *If the event* GoodBox *occurs (as defined in Fig. 3) and the* DCS, Obf *and the puncturable* PRF *schemes are secure, then,*

$$\Pr[R(\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}) = b^*] \geq \frac{1}{2} + \epsilon - \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,0}$ *are as defined in line 2(a)ii of the* Trace *algorithm.*

Proof. We first recall the definition of the event GoodBox: $\mathcal{I} \cap \mathcal{J} = \emptyset$, $C(x_{\mathcal{J}}) = 0$, $C(x_{\mathcal{I}}) = 0$ and R is a minimally-useful box w.r.t. $(C, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{pp}, s_0, s_1, b, \epsilon)$, where $\mathsf{sh}_i = (i, \mathsf{ct}_i, \sigma_i)$ are the honestly-generated shares of parties in $\mathcal{J}$.

By definition 4.4 and the fact that $\mathcal{J} \cap \mathcal{I} = \emptyset$, if we sample fresh shares of parties in $\mathcal{I}$ which are identically distributed as honest shares, then the box must output b with probability $\frac{1}{2} + \epsilon$. Formally,

$$\Pr[R(\{\mathsf{sh}'_i\}_{i\in\mathcal{I}}) = b] \geq \frac{1}{2} + \epsilon$$

where, for all $i \in \mathcal{I}$, sh'_i is sampled as follows: $\mathsf{ct}'_i \leftarrow\$ \mathsf{Enc}(k, (0,0,0))$ and $\sigma'_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}'_i))$. Note that $\mathcal{J} \cap \mathcal{I}$ is crucial for this step, since if there was a party in $\mathcal{J} \cap \mathcal{I}$ then we cannot sample a fresh share independent of its original share sh_i.

Let us use H_0 to refer to this original game, wherein the challenger runs R with randomly sampled shares of $\mathcal{I}$ encrypting $(0,0,0)$, and the box outputs b with some probability. We will now define a series of hybrids:

- H_1: Here, we change how we sample the keys of the doubly-constrained signature DCS. Specifically, the challenger first samples the ciphertexts for all parties in $\mathcal{J} \cup \mathcal{I}$. Then, instead of running KeyGen as in the honest Share algorithm, the challenger samples a constrained key-pair, by running ConstPK. The input circuit C_1 is the same as that defined in line 1 of the Share algorithm, and the circuit C_2 is defined as follows: $C_2(i, \mathsf{ct}''_i)$ is 1 if and only if, (i) $i \in \mathcal{J} \cup \mathcal{I}$ and (ii) $\mathsf{ct}''_i = \mathsf{ct}_i$ for $i \in \mathcal{J}$ and $\mathsf{ct}''_i = \mathsf{ct}'_i$ for $i \in \mathcal{I}$, i.e., the ciphertext must match the ones sampled by the challenger. Note that C_1 is not a subset of C_2 as defined, but that does not affect our security proof since the adversary in this game never sees the tracing key.

  We claim that if the box's behavior differs by a non-negligible amount from H_0 to H_1, we can use it to break security of the DCS scheme. To see why, observe that the view of the adversary only includes the obfuscated program which contains the public key pk, and honestly-generated shares of $\mathcal{J} \cup \mathcal{I}$ – each of these shares contains a signature on messages which satisfy C_2. Hence, we can make an adversary $\mathcal{B}_1$ that acts as the challenger to $\mathcal{A}$, and uses it to win the DCS security game. Specifically, $\mathcal{B}_1$ gets as input $(C, \mathcal{J}, \mathcal{I}, s_0, s_1)$ from $\mathcal{A}$ (it gets $\mathcal{I}$ because we're considering the semi-adaptive tracing game). It samples a PRF key k, and computes ciphertexts encrypting $(0,0,0)$ for shares of all parties in $\mathcal{J} \cup \mathcal{I}$, i.e., $\mathsf{ct}_i \; \forall \; i \in \mathcal{J}$ and $\mathsf{ct}'_i \; \forall \; i \in \mathcal{I}$.

  It then forwards to its challenger (ℓ, z, C_1, C_2), wherein C_1 is as defined in Line 1, and C_2 is as defined above, and gets back a public key pk. To generate shares sh_i for $\mathcal{J}$, it queries the SignO oracle to get a signature on (i, ct_i). It samples b randomly from $\{0,1\}$, obfuscates program P containing pk, s_b, as defined in Fig. 8, and sends to $\mathcal{A}$ the obfuscated program along with $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$.

  It then gets a box R from $\mathcal{A}$. It generates shares sh'_i for all $i \in \mathcal{I}$ by also encrypting $(0,0,0)$, and by querying the SignO oracle to get signatures on (i, ct'_i). It runs R with the shares $\{\mathsf{sh}'_i\}_{i\in\mathcal{I}}$. If R outputs b then it sends 0 to its challenger, otherwise it sends 1. Clearly, $\mathcal{B}_1$ perfectly simulates the two hybrids to $\mathcal{A}$ – H_0 (and H_1) corresponds to $b = 0$ (and $b = 1$) in the DCS security game. Hence, we get that R must still output b with probability $\geq \frac{1}{2} + \epsilon - \mathsf{Adv}^{\mathsf{sec}}_{\mathcal{B}_1,\mathsf{DCS}}(\lambda)$.

- H_2: We now change the program P as follows. We hard-code the sets $\mathcal{J}$ and $\mathcal{I}$ in the program, along with the shares $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$ and $\{\mathsf{sh}'_i\}_{i\in\mathcal{I}}$. Moreover, before checking any signatures, we first set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$, and for input shares of parties in $(\mathcal{J} \cup \mathcal{I})$, we remove them from $\hat{J}$ if their input does not match the hard-coded share. Figure 11 shows the full program.

  We argue that this change does not alter the functionality of the program. To see that, observe that there are no valid signatures on parties outside of $(\mathcal{J} \cup \mathcal{I})$, so these parties would have been excluded from $\hat{J}$ anyways. Similarly, for parties in $(\mathcal{J} \cup \mathcal{I})$, there exist valid signatures only for the shares hard-coded in the box. Hence, the resulting $\hat{J}$ is not affected for any input to the program. More formally, this means that if the box behaves differently in hybrids H_1 and H_2, we can use $\mathcal{A}$

to break security of obfuscation. Hence, R must output b in this hybrid with probability $\geq \frac{1}{2} + \epsilon - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1,\mathsf{DCS}}(\lambda) - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_2,\mathsf{Obf}}(\lambda)$.

$\underline{\mathsf{P}(\{i, \mathsf{ct}_i, \sigma_i\}_{i\in J})}$:

1. **Constants**: C, k, pk, s and $\mathcal{J}, \mathcal{I}$, $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$ and $\{\mathsf{sh}'_i\}_{i\in\mathcal{I}}$.
2. Parse ct_i as (r, ct'_i).
3. Let $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$.
4. For all $i \in \mathcal{J} \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
5. For all $i \in \mathcal{I} \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}'_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
6. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
7. For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
8. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1, \ldots, n\}$. Compute $S \leftarrow \bigcap_{i\in\hat{J}} S_i$.
9. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
10. Set $\hat{J} = \hat{J} \cap S$.
11. Abort if $C(x_{\hat{J}}) = 0$.
12. If $\forall\, i \in \hat{J}$, $f_i = 0$, then output s.
13. Otherwise, output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

Figure 11: The modified program used in hybrid H_2 in the $\emptyset$-Strong traceability proof for our scheme iOGTSS.

- H_3: In hybrid H_3, we again change the program P. Specifically, we remove the last line of the program (which outputs m_{i^*} for some $i^* \in \hat{J}$) and the program simply outputs s. Figure 12 depicts the full program. We argue that this does not change functionality of the program. To see why, we first observe that for any inputs to the program in H_2, the resulting $\hat{J}$ must be a subset of $\mathcal{J} \cup \mathcal{I}$, and the corresponding shares must be the hard-coded shares, each of which encrypt $(0, \cdot, \cdot)$. This means, that in the last line, the program always outputs s. Hence, the modified program in this hybrid is still functionally equivalent. If the box behaves differently in hybrids H_2 and H_3, we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}_3$ that breaks security of obfuscation. We get, R must output b in this hybrid with probability $\geq \frac{1}{2} + \epsilon - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1,\mathsf{DCS}}(\lambda) - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_2,\mathsf{Obf}}(\lambda) - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_3,\mathsf{Obf}}(\lambda)$.
- H_4: We modify the obfuscated program again: We remove lines 6 to 10. In particular, we do not verify the signatures in the input shares, and also do not decrypt the ciphertexts in those shares. We argue that this does not change the functionality of the program. To see why, observe that for any input set J, we know that $\hat{J}$ is set to $J \cap (\mathcal{J} \cup \mathcal{I})$, and even for the remaining parties in J, the program only keeps the shares that match the hard-coded shares. This means that the resulting $\hat{J}$ (even before line 6) is only a subset of $\mathcal{J} \cup \mathcal{I}$, and the corresponding input shares match the hard-coded shares, which contain an encryption to $(0, 0, 0)$ along with valid signatures. Hence, the resulting $\hat{J}$ at the end of line 10 in H_3 is indeed the same as that before line 6.
- H_5: We now puncture the PRF key k. Let $\mathsf{sh}_i = (i, (r_i, \hat{\mathsf{ct}}_i), \sigma_i)$ for all $i \in \mathcal{J}$, and $\mathsf{sh}'_i = (i, (r'_i, \hat{\mathsf{ct}}'_i), \sigma'_i)$ for all $i \in \mathcal{I}$. We puncture k at the set $T := \{r_i\}_{i\in\mathcal{J}} \cup \{r'_i\}_{i\in\mathcal{I}}$. Note that we do not need to hard-code $z_r := \mathsf{PRF.F}(k, r)$ in the program since it never has to decrypt these shares. Figure 13 depicts

$\underline{\mathsf{P}(\{i, \mathsf{ct}_i, \sigma_i\}_{i \in J})}$:

1. **Constants**: C, k, pk, s and $\mathcal{J}, \mathcal{I}$, $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$ and $\{\mathsf{sh}'_i\}_{i \in \mathcal{I}}$.
2. Parse ct_i as (r, ct'_i).
3. Let $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$.
4. For all $i \in \mathcal{J} \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
5. For all $i \in \mathcal{I} \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}'_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
6. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
7. For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
8. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1, \ldots, n\}$. Compute $S \leftarrow \bigcap_{i \in \hat{J}} S_i$.
9. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
10. Set $\hat{J} = \hat{J} \cap S$.
11. Abort if $C(x_{\hat{J}}) = 0$.
12. Output s.

Figure 12: The modified program used in hybrid H_3 in the $\emptyset$-Strong traceability proof for our scheme iOGTSS.

the program. H_5 should be indistinguishable from H_4 to $\mathcal{A}$ by security of obfuscation, since the functionality is identical.

- H_6: We can now replace each $\hat{\mathsf{ct}}_i$ for $i \in \mathcal{J}$ and $\hat{\mathsf{ct}}'_i$ for $i \in \mathcal{I}$ with a uniformly random value. This is indistinguishable from the previous hybrid by the security of punctured PRFs. This now means that by one-time pad security, if the r_i used in all ciphertexts for $\mathcal{J} \cup \mathcal{I}$ are unique, then, each ciphertext $\{\hat{\mathsf{ct}}_i\}_{i \in \mathcal{J}}$ and $\{\hat{\mathsf{ct}}'_i\}_{i \in \mathcal{I}}$ is now uniformly random, except with probability $\leq n^2/2^\lambda$.

Now, we will consider a different hybrid, $\hat{H}_0$, wherein we run the box R with shares generated by the tracer, i.e., $\{\mathsf{sh}^*_{i,0}\}_{i \in \mathcal{I}}$. Each $\mathsf{sh}^*_{i,0}$ contains a ciphertext $\mathsf{ct}^*_{i,0}$ which encrypts $(1, 0, s_{b^*})$, and a signature $\sigma^*_{i,0}$ on $(i, \mathsf{ct}^*_{i,0})$ generated using the constrained key sk_1. Let us use p^* to denote the probability that R outputs b^* given these inputs. We now define a few hybrids:

- $\hat{H}_1$: We change the signatures $\sigma^*_{i,0}$ to be generated by the unconstrained signing key sk instead of sk_1. Because these signatures are on messages that satisfy the constraint C_1, and by the identical-distribution property of DCS, $\hat{H}_1$ is computationally indistinguishable from $\hat{H}_0$. More formally, if the behavior of the adversary differs by a non-negligible amount between $\hat{H}_0$ and $\hat{H}_1$, then we can use it to construct an adversary $\hat{\mathcal{B}}$ to break identical distribution property of DCS. Formally, $\hat{\mathcal{B}}$ gets $C, \mathcal{J}, \mathcal{I}, s_0, s_1$ from $\mathcal{A}$. It then samples a PRF key k, and constructs circuit C_1 as described in Fig. 8. It sends C_1 to its challenger, and gets a public key pk. It uses pk to construct the program P (by sampling $s \leftarrow\!\!\$\ \{s_0, s_1\}$). For generating shares of any party i in $\mathcal{J}$, it samples ct_i as an encryption of $(0, 0, 0)$ and queries its challenger for a signature on (i, ct_i). It then sends these shares along with pp to $\mathcal{A}$. Lastly, it receives a reconstruction box R from $\mathcal{A}$. For all parties i in $\mathcal{I}$, it samples $\mathsf{ct}^*_{i,0}$ as encryptions of $(1, 0, s_{b^*})$ (where $b^* \leftarrow\!\!\$\ \{0, 1\}$) and queries its challenger for a signature on $(i, \mathsf{ct}^*_{i,0})$. It runs R with these shares and forwards the output to its challenger. It is clear that $\hat{\mathcal{B}}$ perfectly simulates the game to $\mathcal{A}$. Moreover, $b = 0$ in the id-dist game corresponds to hybrid $\hat{H}_0$ and $b = 1$

$\underline{\mathsf{P}(\{i, \mathsf{ct}_i, \sigma_i\}_{i \in J})}$:

1. **Constants**: C, k^T, pk, s and $\mathcal{J}, \mathcal{I}$, $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$, $\{\mathsf{sh}'_i\}_{i \in \mathcal{I}}$, $\{z_r\}_{r \in T}$.
2. Parse ct_i as (r, ct'_i).
3. Let $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$.
4. For all $i \in \mathcal{J} \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
5. For all $i \in \mathcal{I} \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}'_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
6. Abort if $C(x_{\hat{J}}) = 0$.
7. Output s.

Figure 13: The modified program used in hybrid H_5 in the $\emptyset$-strong traceability proof for our scheme iOGTSS.

corresponds to $\hat{H}_1$. Hence, if the probability with which R outputs b^* in $\hat{H}_1$ differs by more than a negligible amount, then $\hat{\mathcal{B}}$ wins its game with non-negligible advantage.

- $\hat{H}_2$: Similar to H_1 above, we generate a constrained key-pair for the doubly-constrained signature. We run ConstPK with C_1 as defined in line 1 of the Share algorithm, and the circuit C_2 outputs 1 on input (i, ct''_i) if and only if, (i) $i \in \mathcal{J} \cup \mathcal{I}$ and (ii) $\mathsf{ct}''_i = \mathsf{ct}_i$ for $i \in \mathcal{J}$ and $\mathsf{ct}''_i = \mathsf{ct}^*_{i,0}$ for $i \in \mathcal{I}$, i.e., the ciphertext must match the ones sampled by the challenger. By a similar argument as given in H_1, R must output b^* in this hybrid with almost the same probability as in $\hat{H}_1$, by DCS security.
- $\hat{H}_3$: Similar to H_2, we hard-code the set $\mathcal{J} \cup \mathcal{I}$ and the corresponding shares in the program.
- $\hat{H}_4$: Similar to H_3 above, we change the program P – we remove s, hard-code $s^* = s_{b^*}$ instead and change the last line to simply output s^*. We argue that this does not change the functionality of the program. To see why, observe that for any inputs to P, the resulting $\hat{J}$ must be a subset of $\mathcal{J} \cup \mathcal{I}$, and the corresponding shares must be the hard-coded shares. Next, if the resulting $\hat{J}$ is a subset of $\mathcal{J}$, then the program will abort because $C(x_{\mathcal{J}}) = 0$. On the other hand, if $\hat{J}$ has at least one share from $\mathcal{I}$, lets say $\mathsf{sh}^*_{i,0}$, observe that this share encrypts $(1, \cdot, s^*)$, which means that the program will output s^* for any such input. In other words, it will never output s and hence this hybrid is indistinguishable from $\hat{H}_3$ by iO security.
- $\hat{H}_5$: Similar to H_4, we now remove the logic in the program that checks the signatures and decrypts the ciphertexts in the input shares. By a similar argument as H_4, this does not impact the functionality of the program since the shares of $\hat{J}$ anyway are the valid shares encrypting $(0, 0, 0)$ for $\mathcal{J}$ and $(1, 0, s_{b^*})$ for $\mathcal{I}$, that are hard-coded in the program – these shares have valid signatures, and the resulting set $\hat{J}$ also remains unchanged.
- $\hat{H}_6$: Similar to H_5, we now puncture the PRF key k at the set T as defined above. Again, we do not hard-code the corresponding PRF values z_r in the program since the ciphertexts in shares of $\mathcal{J} \cup \mathcal{I}$ are anyway not decrypted. By a similar argument as H_5, this is indistinguishable from $\hat{H}_5$ by iO security.
- $\hat{H}_7$: We can now replace the ciphertexts in shares of $\mathcal{J} \cup \mathcal{I}$ with a uniformly random value, as in hybrid H_6. This is indistinguishable from the previous hybrid by security of punctured PRFs.

  Now, we argue that the hybrids H_6 and $\hat{H}_7$ are indistinguishable to the adversary. The two things that differ in these two hybrids are (i) the last line of the program P outputs s in H_6 but s^* in $\hat{H}_7$,

(ii) the ciphertexts for the set $\mathcal{I}$ encrypt $(0, \cdot, 0)$ in H_5 and $(1, \cdot, s^*)$ in $\hat{H}_6$. But observe that, both s and s^* come from the same distribution, i.e., they are both sampled uniformly at random from $\{s_0, s_1\}$. Moreover, the ciphertexts in the shares of $\mathcal{I}$ are distributed uniformly at random as well, by one-time pad security as argued in both hybrids. Hence, the behavior of R must be identical in the two hybrids. This proves that R must output b^* with probability $\frac{1}{2} + \epsilon - \mathsf{negl}(\lambda)$ where the negligible term comes from the security of DCS, obfuscation and the punctured PRF.

□

Next, Lemma 5.8 below proves that the box cannot output b^* with probability more than $1/2$ when given shares with ciphertexts encrypting $(1, n, s^*)$.

Lemma 5.8. *If* GoodBox *occurs then,*

$$\Pr[R(\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}) = b^*] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,n}$ *shares are as defined in line 2(a)ii of the* Trace *algorithm.*

Proof. We make a series of hybrids. Let H_0 denote the original $\emptyset$-Strong traceability game, wherein the box R is given as inputs $\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}$. Let $\mathsf{sh}^*_{i,n} = (i, \mathsf{ct}^*_{i,n}, \sigma^*_{i,n})$ for all $i \in \mathcal{I}$ where $\sigma^*_{i,n}$ is generated using the constrained key sk_1.

We first define a hybrid $\hat{H}_0$, wherein the signatures $\sigma^*_{i,n}$ are now generated by the unconstrained signing key sk instead of sk_1. Because these signatures are on messages that satisfy the constraint C_1, and by the identical-distribution property of DCS, $\hat{H}_0$ is computationally indistinguishable from H_0. If not, we can use $\mathcal{A}$ to construct an adversary $\hat{\mathcal{B}}$ to break identical distribution property of DCS, as described in hybrid $\hat{H}_1$ in proof of Lemma 5.7.

We define a new hybrid H_1, where we change how we sample the keys of the doubly-constrained signature scheme DCS. Instead of running KeyGen as in the honest Share algorithm, the challenger runs ConstPK to generate a constrained key-pair. The circuit C_1 is as defined in the Share algorithm, and C_2 is constructed in the same way as in hybrid H_1 in the proof of Lemma 5.7. Specifically, $C_1(i, \mathsf{ct}''_i)$ is 1 only if $\mathsf{Dec}(k, \mathsf{ct}''_i) = (1, \cdot, \cdot)$. $C_2(i, \mathsf{ct}''_i)$ is 1 if and only if (i) $i \in \mathcal{J} \cup \mathcal{I}$ and (ii) $\mathsf{ct}''_i = \mathsf{ct}_i$ if $i \in \mathcal{J}$, and $\mathsf{ct}''_i = \mathsf{ct}^*_{i,n}$ if $i \in \mathcal{I}$, i.e., the ciphertext must match the ones sampled by the challenger. Moreover, all the signatures are now generated using sk_{C_2}. Note that C_1 is not a subset of C_2 but that does not affect this proof since the adversary never sees the constrained signing key sk_{C_1}.

Similar to hybrid H_1 in the proof of Lemma 5.7, if the box behaves differently in H_1 than in H_0, we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks security of the DCS scheme.

Next, in hybrid H_2, we change the program P, similar to hybrid H_2 in proof of Lemma 5.7. We hard-code the sets $\mathcal{J}$ and $\mathcal{I}$ and their corresponding shares $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}$. Moreover, before checking any signatures, we first set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$, and for input shares of parties in $(\mathcal{J} \cup \mathcal{I})$, we remove them from $\hat{J}$ if their input does not match the hard-coded share. The full program is similar to Figure 11, with shares sh'_i replaced with $\mathsf{sh}^*_{i,n}$. By an argument similar to that in Lemma 5.7, the functionality of the program is unchanged, and so the box must behave similarly in hybrids H_1 and H_2.

In hybrid H_3, we modify the program again – we simply remove the last two lines, and remove s from the program. We argue that this does not change the functionality of the program. To see that, observe that the resulting set $\hat{J}$ in the program in H_2 is always a subset of $\mathcal{J} \cup \mathcal{I}$. Moreover, if $\hat{J}$ is a subset of $\mathcal{J}$, then the program would abort anyway since $C(x_{\mathcal{J}}) = 0$. So we only need to worry about inputs such that the resulting $\hat{J}$ has at least one share from $\mathcal{I}$. But for any party in $\mathcal{I}$, the only valid share is $\mathsf{sh}^*_{i,n} = (i, \mathsf{ct}^*_{i,n}, \sigma^*_{i,n})$ (since there is no valid signature on any other ciphertext), where $\mathsf{ct}^*_{i,n}$ decrypts to $(1, n, s^*)$. This means that the set S will end up being a subset of $\mathcal{I}$, and as a result, $\hat{J} \subseteq \mathcal{I}$. But since

$C(x_{\mathcal{I}}) = 0$, the program will always abort for these inputs as well. Hence, it never reaches the last lines, and so the functionality of the modified program is the same as that in H_2.

In hybrid H_4, we change the program again, wherein for any $i \in \mathcal{I} \cup \mathcal{J}$, we do not run Dec on ct_i (and consequently, do not compute S and simply abort). We argue that this does not change the functionality of the program. This is because, the program aborts regardless of the decryption value of these ciphertexts. Hence, hybrids H_3 and H_4 are indistinguishable by iO security.

In hybrid H_5, we puncture the PRF key k at the set $T = \{r_i\}_{i\in\mathcal{J}} \cup \{r_i^*\}_{i\in\mathcal{I}}$, where $\mathsf{sh}_i = (i, (r_i, \hat{\mathsf{ct}}_i), \sigma_i)$ for all $i \in \mathcal{J}$ and $\mathsf{sh}^*_{i,n} = (i, (r_i^*, \hat{\mathsf{ct}}^*_{i,n}), \sigma^*_{i,n})$ for all $i \in \mathcal{I}$. This is indistinguishable from hybrid H_4 since the program does not need to query PRF at values in T.

In hybrid H_6, we modify the ciphertexts for all $i \in \mathcal{J}$ to be (r_i, z_i) where $z_i \leftarrow\!\$\ \{0,1\}^{1+\log(n+1)+\log(|\mathcal{S}_\lambda|)}$ is sampled uniformly at random, similarly the ciphertexts for all $i \in \mathcal{I}$ are modified to be (r_i^*, z_i^*) where $z_i \leftarrow\!\$\ \{0,1\}^{1+\log(n+1)+\log(|\mathcal{S}_\lambda|)}$. This is indistinguishable from hybrid H_5 by security of punctured PRFs.

But now observe that in hybrid H_6, the view of the adversary contains no information about b^*, and hence, the probability of R outputting b^* cannot be more than $1/2$. This proves the lemma. □

Let $\hat{p}_j$ denote the probability with which the box R outputs b^* when run with shares $\{\mathsf{sh}^*_{i,j}\}_{i\in\mathcal{I}}$. Lemma 5.7 proves that $\hat{p}_0 = \frac{1}{2} + \epsilon - \mathsf{negl}(\lambda)$. Lemma 5.8 proves that $\hat{p}_n \le \frac{1}{2} + \mathsf{negl}(\lambda)$. Combining the two, we get that there must be a jump in these probabilities for some $j \in [n]$. Formally, let $\hat{\epsilon} = \hat{p}_0 - \hat{p}_n \approx \epsilon$. Then, there exists at least one $j^* \in [n]$ such that, $\hat{p}_{j^*-1} - \hat{p}_{j^*} \ge \epsilon/2n$.

Next, observe that the values p'_j computed in the Trace algorithm are an approximation of the probability $\hat{p}_j$. By Chernoff bound, for all $j \in \{0, \ldots, n\}$,

$$\Pr[|\hat{p}_j - p'_j| > \frac{\epsilon}{16n}] < 2e^{-\lambda/16}.$$

Hence, we get that, for j^*, $p'_{j^*-1} - p'_{j^*} \ge \epsilon/2n - \epsilon/8n > \epsilon/4n$. This marks the end of the first part, which proves that if the event GoodBox occurs, then the Trace algorithm will blame at least one party i.e., j^*. Hence, $\mathcal{J}^* \neq \perp$, except with negligible probability in the event GoodBox.

We now move to the second part, where we prove that the jump can not be honest. Crucially, this holds even if the adversary corrupts upto $n-1$ parties, and chooses arbitrary label $\mathcal{I}$, and does not output a minimally-useful box. We prove this via two lemmas.

Lemma 5.9. *Let $\mathcal{J}, \mathcal{I}$ and $\{\hat{p}_j\}_{j\in\{0,1,\ldots,n\}}$ be as defined above. For all $j \in [n] \setminus (\mathcal{J} \cup \mathcal{I})$,*

$$|\hat{p}_j - \hat{p}_{j-1}| \le \mathsf{negl}(\lambda).$$

Proof. Let H_0 denote the original game where the challenger honestly runs Share, and runs the box R with $\{\mathsf{sh}^*_{i,j-1}\}_{i\in\mathcal{I}}$ where $\mathsf{sh}^*_{i,j-1}$ are generated by the tracer. For all $i \in \mathcal{I}$, let $\mathsf{sh}^*_{i,j-1} = (i, \mathsf{ct}^*_{i,j-1}, \sigma^*_{i,j-1})$.

We define a hybrid $\hat{H}_0$, wherein the signatures $\sigma^*_{i,j-1}$ in $\mathsf{sh}^*_{i,j-1}$ are generated using the unconstrained signing key sk instead of sk_1, for all $i \in \mathcal{I}$. By a similar argument as for $\hat{H}_0$ in Lemma 5.8, this hybrid is computationally indistinguishable from H_0, based on the security of DCS.

We define another hybrid H_1, wherein the challenger samples a constrained key-pair for the DCS scheme. Specifically, it first samples the PRF key k, $b^* \leftarrow\!\$\ \{0,1\}$, and ciphertexts for shares of parties in $\mathcal{J}$ i.e., ct_i encrypting $(0,0,0)$ and ciphertexts $\mathsf{ct}^*_{i,j-1}$ for parties in $\mathcal{I}$ encrypting $(1, j-1, s^*)$ where $s^* = s_{b^*}$. It then runs ConstPK instead of KeyGen, with inputs ℓ, z and $C_1 = 1$ if Dec outputs $(1, \cdot, \cdot)$, as in the Share algorithm, and C_2 defined as follows: $C_2(i, \mathsf{ct}''_i)$ outputs 1 if, (i) $i \in \mathcal{J} \cup \mathcal{I}$ and ct''_i matches the ciphertexts sampled by the challenger for i, or (ii) $i \neq j$. In other words, there are no valid signatures for party j with respect to the constrained key. Note that C_1 is not a subset of C_2 as defined, but that does not affect our proof since the adversary in this game never sees the tracing key.

We claim that if the adversary can distinguish between hybrids $\hat{H}_0$ and H_1, we can use it to make an adversary $\mathcal{B}$ that breaks security of the DCS scheme. To see why, observe that the view of the adversary only includes the obfuscated program which contains the public key pk, and honestly-generated shares of $\mathcal{J} \cup \mathcal{I}$ – each of these shares contains a signature on messages which satisfy C_2. Hence, $\mathcal{B}$ can act as the challenger to $\mathcal{A}$, and use it to win the DCS security game.

Specifically, $\mathcal{B}$ gets as input $(C, \mathcal{J}, \mathcal{I}, s_0, s_1)$ from $\mathcal{A}$ (it gets $\mathcal{I}$ because we're considering the semi-adaptive tracing game). It samples a PRF key k, bit $b^* \leftarrow\$ \{0,1\}$, and computes ciphertexts ct_i encrypting $(0,0,0)$ for shares of all parties in $i \in \mathcal{J}$, and ciphertexts $\mathsf{ct}^*_{i,j-1}$ encrypting $(1, j-1, s^*)$ for all parties $i \in \mathcal{I}$, where $s^* = s_{b^*}$.

It then forwards to its challenger (ℓ, z, C_1, C_2), wherein $C_1 = 1$ if Dec outputs $(1, \cdot, \cdot)$, and C_2 is as defined above, and gets back a public key pk. To generate shares sh_i for $\mathcal{J}$, it queries the SignO oracle to get a signature on (i, ct_i). It samples b randomly from $\{0,1\}$, obfuscates program P containing pk, s_b, as defined in Fig. 8, and sends to $\mathcal{A}$ the obfuscated program along with $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$.

It then gets a box R from $\mathcal{A}$. It generates shares $\mathsf{sh}^*_{i,j-1}$ for all $i \in \mathcal{I}$, by querying the SignO oracle to get signatures on $(i, \mathsf{ct}^*_{i,j-1})$. It runs R with the shares $\{\mathsf{sh}^*_{i,j-1}\}_{i \in \mathcal{I}}$. If R outputs b then it sends 0 to its challenger, otherwise it sends 1. Clearly, $\mathcal{B}$ perfectly simulates the two hybrids to $\mathcal{A}$ – $\hat{H}_0$ (and H_1) corresponds to $b = 0$ (and $b = 1$) in the DCS security game. Hence, the behavior of the box in hybrids H_1 and $\hat{H}_0$ must only differ by a negligible amount.

Next, we define H_2 where we change the program P. Specifically, we hard-code $\mathcal{J} \cup \mathcal{I}$ and their corresponding ciphertexts, along with j in the program. Before checking any signatures, we simply set $\hat{J} \leftarrow \hat{J} \setminus \{j\}$. Moreover, for input shares of parties in $\mathcal{J} \cup \mathcal{I}$, we remove them from $\hat{J}$ if their input does not match the hard-coded ciphertext. We argue that this does not change the functionality of the program. This is because, in H_1, we constrained the public key of DCS so there is no valid signature for party j, which means that it would anyway be removed from $\hat{J}$ when the signatures are verified. Similarly, for parties in $\mathcal{J} \cup \mathcal{I}$, there is no valid signature except on the hard-coded ciphertexts, so these parties would also have been removed if the input ciphertext did not match.

Let us denote $\mathsf{sh}^*_{i,j-1} = (i, (r_i, \hat{\mathsf{ct}}_{i,j-1}), \sigma^*_{i,j-1})$, for all $i \in \mathcal{I}$. In H_3, we puncture the PRF key k at all points in $T = \{r_i\}_{i \in \mathcal{I}}$, and hard-code $z_{r_i} = \mathsf{PRF.F}(k, r_i)$ in the program for all $i \in \mathcal{I}$. To decrypt any ciphertext of the form (r, ct) where $r \in T$, we now compute $\mathsf{ct} - z_r$. This does not change the functionality of the program, so H_3 is indistinguishable from H_2 by iO security.

In H_4, we sample each z_{r_i} value uniformly at random. If the r_i values are different for all shares, then, by one-time pad security, the ciphertexts $\mathsf{ct}^*_{i,j-1}$ are also random. This means we can now change from the encryptions of $(1, j-1, s^*)$ to encryptions of $(1, j, s^*)$ by simply adding $(0||1||0)$ to the corresponding z_{r_i} value, for all $i \in \mathcal{I}$. Observe that the functionality of the program remains the same even with this change – j is removed from $\hat{J}$ in both cases if such a ciphertext is part of the input.

This means that, behavior of the box R cannot differ by more than a negligible amount between shares $\{\mathsf{sh}^*_{i,j-1}\}$ encrypting $(1, j-1, s^*)$ and shares $\{\mathsf{sh}^*_{i,j}\}$ encrypting $(1, j, s^*)$. □

By Chernoff bound, we know that p'_j is within $\epsilon/16n$ of $\hat{p}_j$ for all $j \in \{0, \ldots, n\}$. Hence, Lemma 5.9 implies that for all $j \in [n] \setminus (\mathcal{J} \cup \mathcal{I})$, $p'_{j-1} - p'_j \leq \epsilon/8n$ except with negligible probability. Hence, the Trace algorithm will never output any party in $[n] \setminus (\mathcal{J} \cup \mathcal{I})$.

To finish the proof, in Lemma 5.10, we prove that the Trace will never blame any honest party in $\mathcal{I}$.

Lemma 5.10. *Let $\mathcal{J}, \mathcal{I}$ and $\{\hat{p}_j\}_{j \in \{0,1,\ldots,n\}}$ be as defined above. For all $j \in \mathcal{I} \setminus \mathcal{J}$,*

$$|\hat{p}_j - \hat{p}_{j-1}| \leq \mathsf{negl}(\lambda).$$

Proof. We use a series of hybrids for the proof. Let H_0 refer to the hybrid where the challenger runs Share honestly, and then runs R with inputs $\{\mathsf{sh}^*_{i,j-1}\}_{i \in \mathcal{I}}$. Let $\mathsf{sh}^*_{j,j-1} = (j, \mathsf{ct}^*_{j,j-1}, \sigma^*_{j,j-1})$ where $\mathsf{ct}^*_{j,j-1}$ encrypts $(1, j-1, s^*)$.

We first define a hybrid $\hat{H}_0$, wherein the signatures $\sigma^*_{i,j-1}$ in $\mathsf{sh}^*_{i,j-1}$ are generated using the unconstrained signing key sk instead of sk_1, for all $i \in \mathcal{I}$. By a similar argument as for $\hat{H}_0$ in Lemma 5.8, this hybrid is computationally indistinguishable from H_0.

We define a hybrid H_1, where the challenger samples a constrained key pair for DCS. It first samples ciphertexts for parties in $\mathcal{J} \cup \mathcal{I}$. It then runs $\mathsf{ConstPK}$ instead of KeyGen, with $C_1 = 1$ if Dec outputs $(1, \cdot, \cdot)$ as defined in the Share algorithm, and C_2 which accepts input (i, ct_i) only if either (i) $i \notin \mathcal{I} \cup \mathcal{J}$, or (ii) if $i \in \mathcal{I} \cup \mathcal{J}$ then ct_i equals the ciphertext sampled by the challenger. In particular, the only valid ciphertext for j is the one sampled by the challenger that encrypts $(1, j-1, s^*)$ (since $j \notin \mathcal{J}$). We argue that if the adversary $\mathcal{A}$ behaves differently in the two hybrids, we can use it to break security of the DCS scheme. Similar to H_1 defined in proof of Lemma 5.9, we can make an adversary $\mathcal{B}$ that perfectly simulates the hybrids to $\mathcal{A}$ using its security challenger. This is because, the adversary only sees signatures on messages that satisfy C_2. Hence, the box R must behave identically in the two hybrids except with negligible probability.

Next, in hybrid H_2, we change the program P. We hard-code the sets $\mathcal{J}$ and $\mathcal{I}$ along with their ciphertexts in the program, and before checking any signatures, for any input share for $i \in \mathcal{J} \cup \mathcal{I}$, if the input ciphertext ct_i is not equal to the hard-coded ciphertext, then we remove it from $\hat{J}$. In particular, if party $j \in J$, then it will be removed if the input ciphertext is not $\mathsf{ct}^*_{j,j-1}$. This does not change the functionality because in H_1, we constrained the public key so a valid signature exists for $\mathcal{J} \cup \mathcal{I}$ only for the hard-coded ciphertexts.

Let us denote $\mathsf{sh}^*_{i,j-1} = (i, \mathsf{ct}^*_{i,j-1} = (r_i, \hat{\mathsf{ct}}_{i,j-1}), \sigma^*_{i,j-1})$ for all $i \in \mathcal{I}$. We define another hybrid H_3, where we puncture the PRF key k at the set $T = \{r_i\}_{i \in \mathcal{I}}$. Moreover, we hard-code the PRF evaluations $z_r = \mathsf{PRF.F}(k, r)$ in the program for all $r \in T$, and we use z_r to decrypt any ciphertext of the form $(r, \cdot)$. The functionality of the program remains the same from H_2 to H_3, so the behavior of R must not change except with negligible probability.

In hybrid H_4, we sample all the z_r values uniformly at random. If no two ciphertexts in shares of $\mathcal{I}$ use the same r, then, by one-time pad security, the ciphertexts $\mathsf{ct}^*_{i,j-1}$ for all $i \in \mathcal{I}$ are distributed uniformly random as well.

In hybrid H_5, we will now change the ciphertexts to encrypt $(1, j, s^*)$ for all $i \in \mathcal{I}$, by changing z_r to $z_r + (0||1||0)$. We argue that doing this does not change the functionality of the program. To show that, we consider two types of inputs to the program. Note that since the ciphertexts encrypting j or $j-1$ only affect the resulting $\hat{J}$, we only need to argue that it does not change for any input, when we move from hybrid H_4 to H_5:

- $j \notin J$: The resulting $\hat{J}$ will not have j regardless of whether the ciphertexts encrypt $(1, j, s^*)$ or $(1, j-1, s^*)$, since $\hat{J} \subseteq J$ by construction.
- $j \in J$: The only valid ciphertext for j is the one encrypting $(1, j-1, s^*)$ in hybrid H_4 – since the first bit is 1, j will be in $\hat{J}$ regardless of the other ciphertexts. In hybrid H_5, the only valid ciphertext for j is the one encrypting $(1, j, s^*)$. Again, since the first bit is 1, regardless of the other ciphertexts, j will be in $\hat{J}$. Hence, the resulting set $\hat{J}$ is the same for the two hybrids, even for inputs containing j.

This proves that $\hat{p}_j$ and $\hat{p}_{j-1}$ cannot differ by more than a negligible amount.

□

By a similar analysis as above, by Chernoff bound, p'_j and p'_{j-1} will also differ by a maximum of $\epsilon/8n$, and hence, honest parties in $\mathcal{I}$ will also not be blamed by the tracer. This completes the proof. □

Efficiency. The share size for our scheme is $\mathsf{poly}(\log(n))$. The parameter z is set to be the size of the largest circuit C_1 and C_2 defined across the security proofs – it is bounded by $O(n\log(n))$, and the parameter $\ell = \log(n) + \lambda + 1 + \log(n+1)$. The public information contains the obfuscated program, which needs to be padded to be of length $O(\mathsf{poly}(n))$ because we use programs of this size in our security proofs. Lastly, the tracing key is of size $\mathsf{poly}(\log(n), \lambda)$.

5.3 Re-usability of Public Parameters

In this section, we show how the public parameters of our scheme can be re-used if the same dealer shares multiple secrets. Let us start with defining the syntax for a traceable secret sharing scheme with re-usable public parameters:

Syntax. A traceable secret sharing scheme with re-usable public parameters for a circuit ensemble $\{\mathcal{C}_\lambda\}_\lambda$ is a tuple of five PPT algorithms $\mathsf{rGTSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp_r}, \mathsf{state})$ is the randomized setup algorithm that outputs the re-usable public parameters $\mathsf{pp_r}$, along with secret state state.
- $\mathsf{Share}(1^\lambda, s, C, \mathsf{state}) \to (\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk})$ is the randomized set sharing algorithm that takes as input the security parameter 1^λ, the secret $s \in \mathcal{S}_\lambda$, the access structure in the form of a poly-sized monotone circuit $C : \{0,1\}^n \to \{0,1\}$ such that $C \in \mathcal{C}_\lambda$ along with the state state output by the Setup algorithm. It outputs the secret shares of all parties $\{\mathsf{sh}_i\}$, the secret-specific public information pp, the tracing key tk and the verification key vk.
- The rest of the algorithms are as described in Section 4.

Since the Share algorithm relies on the secret state output by Setup, the public parameters $\mathsf{pp_r}$ can only be re-used across multiple secrets shared by the *same dealer*. Note that this notion is different from *reusable* secret sharing [20] where secret shares of the parties are re-used across multiple secrets.

Security. We require a traceable secret sharing scheme with re-usable public parameters to satisfy correctness and the same notions of security as defined in Section 4, but with a stronger definition. In particular, we allow the adversary in all the games to also query secret shares of all parties for arbitrary secrets and access structures of its choice. We formalize this by providing an oracle shareO to the adversary, which takes as input a secret s and a circuit C representing the access structure, and outputs the shares of all n parties for (s, C). We present the formal definitions of tracer secrecy and $\emptyset$-strong traceability with this modification below. The definitions of secrecy and tracer non-imputability can be extended in the same manner.

Definition 5.11 (Tracer Secrecy with re-usable public parameters)**.** *Let* $\mathsf{rGTSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme with re-usable public parameters for secrets in* $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda\in\mathbb{N}}$*. We say that* rGTSS *satisfies* ***tracer secrecy*** *if for every* $\lambda \in \mathbb{N}$*, any two secrets* $s_0, s_1 \in \mathcal{S}_\lambda$*, every monotone circuit* $C : \{0,1\}^n \to \{0,1\}$ *such that* $C \in \mathcal{C}_\lambda$*, and every subset* $\mathcal{J} \subseteq [n]$ *such that* $C(x_\mathcal{J}) = 0$*, every* PPT *algorithm* $\mathcal{A}$*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{t-sec-re}}_{D,\mathsf{GTSS}}(\lambda) := \left|\Pr\left[\mathbf{ExpTracerSecrecyRe}_{\mathcal{A},\mathsf{rGTSS}}(\lambda) = 1\right] - \frac{1}{2}\right|,$$

where $\mathbf{ExpTracerSecrecyRe}_{\mathcal{A},\mathsf{rGTSS}}$ *is as defined in Figure 14.*

Definition 5.12 ($\emptyset$-Strong Traceability with re-usable public parameters)**.** *Let* $\mathsf{rGTSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Rec}, \mathsf{Trace}, \mathsf{Verify})$ *be a traceable secret-sharing scheme with re-usable public parameters, and let* $\epsilon = \epsilon(\lambda)$ *be a*

Experiment $\mathbf{ExpTracerSecrecyRe}_{\mathcal{A},\mathsf{rGTSS}}$

1 : $b \leftarrow\$ \{0,1\}$

2 : $(\mathsf{pp}_\mathsf{r}, \mathsf{state}) \leftarrow\$ \mathsf{Setup}(1^\lambda)$

3 : $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\$ \mathsf{Share}(1^\lambda, s_b, C, \mathsf{state})$

4 : $b^* \leftarrow\$ \mathcal{A}^{\mathsf{shareO}(\cdot,\cdot)}(\mathsf{state}, \mathsf{pp}_\mathsf{r}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{tk}, \mathsf{vk})$

5 : **return** $(b^* == b)$

Oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$

1 : **return** $(\{\mathsf{sh}_j^{(i)}\}_{j\in[n]}, \mathsf{pp}^{(i)}, \mathsf{tk}^{(i)}, \mathsf{vk}^{(i)}) \leftarrow\$ \mathsf{Share}(1^\lambda, s^{(i)}, C^{(i)}, \mathsf{state})$

Figure 14: The tracer secrecy experiment for a traceable secret-sharing scheme rGTSS with re-usable public parameters.

function of the security parameter. We say that GTSS *is* ***traceable****, if for every probabilistic polynomial time adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}^{\emptyset-\mathrm{str}-\mathrm{trac}-\mathrm{re}}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) := \Pr\left[\mathbf{Exp}\emptyset\mathbf{StrongTraceRe}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) = 1\right],$$

where $\mathbf{Exp}\emptyset\mathbf{StrongTraceRe}_{\mathcal{A},\mathsf{GTSS},\epsilon}$ *is defined in Figure 15.*

Our construction with re-usable public parameters. We now show how to modify our construction in Section 5.2 to make the obfuscated program re-usable.

Recall that the obfuscated program has the circuit C and the secret s hard-coded along with the PRF key k and the DCS public key pk. Our key observation is that instead of hard-coding s and C in the program, it can take these values as an additional input. We now sample another PRF key k_s and hard-code it into the program (in addition to k and pk). Next, when sharing any secret s for access structure C, we will encrypt s using k_s and output the resulting ciphertext along with C as secret-specific public parameters. The program will take these as input along with the shares of a subset J, and output the decrypted secret (in place of the hard-coded secret) if all the checks pass. With this change, the size of the obfuscated program is now amortized across multiple secret sharings.

Unfortunately, this modification breaks traceability, because it enables a *mix-and-match* attack. In particular, an adversary can run the obfuscated program using shares corresponding to one secret together with the encryption and public parameters of a different secret. This allows the adversary to trivially distinguish tracer-generated shares as follows. The adversary first requests the real shares and public parameters pp for an arbitrary secret s, and constructs a box containing these values. Given any input shares, the box executes the obfuscated program using the input shares together with the fixed real shares and pp for s. Clearly, if the input shares are tracer-generated, the program outputs the tracer's embedded secret s_{b^*} rather than s. Thus, the box can directly detect whether the input shares came from the tracer, violating traceability.

To prevent such mix-and-match attacks, we modify the signature contained in each secret share to also sign the secret-specific public parameters pp. This binds each share to its corresponding public parameters and prevents combining shares from one secret with the parameters of another. We present the full scheme in Figure 16.

Correctness. Correctness of this scheme follows directly from correctness of the doubly-constrained signature and obfuscation.

Experiment $\mathbf{Exp}\emptyset\mathbf{StrongTraceRe}_{\mathcal{A},\mathsf{rGTSS},\epsilon}(\lambda)$

1 : $(C, \mathcal{J}, s_0, s_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$ // $\mathcal{A}$ chooses the parties to corrupt
2 : $b \leftarrow\!\$ \{0,1\}$
3 : $(\mathsf{pp}_\mathsf{r}, \mathsf{state}') \leftarrow\!\$ \mathsf{Setup}(1^\lambda)$
4 : $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\$ \mathsf{Share}(1^\lambda, s_b, C, \mathsf{state}')$
5 : $(R, \mathcal{I}) \leftarrow\!\$ \mathcal{A}^{\mathsf{shareO}(\cdot,\cdot)}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}})$
6 : $(\mathcal{J}^*, \pi) \leftarrow\!\$ \mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I})$
7 : $\mathsf{GoodBox} := (\mathcal{I} \cap \mathcal{J} = \emptyset) \wedge C(x_\mathcal{J}) = 0 \wedge C(x_\mathcal{I}) = 0 \wedge$
8 : $\quad (R \text{ is } (C, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{pp}, , s_0, s_1, b, \epsilon)\text{-minimally-useful})$
9 : **if** $\mathcal{J}^* \not\subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 1
10 : **if** $\neg\mathsf{GoodBox}$ **then return** 0
11 : **if** $\mathcal{J}^* \neq \emptyset \wedge \mathcal{J}^* \subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 0 **else return** 1

Oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$

1 : $(\{\mathsf{sh}_j^{(i)}\}_{j\in[n]}, \mathsf{pp}^{(i)}, \mathsf{tk}^{(i)}, \mathsf{vk}^{(i)}) \leftarrow\!\$ \mathsf{Share}(1^\lambda, s^{(i)}, C^{(i)}, \mathsf{state}')$
2 : **return** $(\{\mathsf{sh}_j^{(i)}\}_{j\in[n]}, \mathsf{pp}^{(i)})$

Figure 15: The $\emptyset$-strong tracing experiment for a traceable secret-sharing scheme rGTSS with re-usable public parameters and an adversary $\mathcal{A}$.

Tracer Secrecy. We directly prove tracer secrecy since it also implies secrecy. Similar to the scheme in Section 5.2, we prove tracer secrecy of our scheme based on security of the doubly-constrained signature scheme, the PRF and the indistinguishability obfuscation.

Theorem 5.13. *For all* PPT *adversaries* D*, there exist* PPT *adversaries* D_1, D_2, D_3, D_4, D_5*, such that,*

$$\mathsf{Adv}^{\mathrm{t-sec-re}}_{D,\mathsf{iOrGTSS}}(\lambda) \leq \frac{q}{2^\lambda} + \mathsf{Adv}^{\mathrm{sec}}_{D_1,\mathsf{DCS}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_2,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_3,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_4,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{D_5,\mathsf{PRF}}(\lambda),$$

where $q = q(\lambda)$ *is an upper bound on the number of* shareO *queries made by the adversary.*

Proof. We consider a series of hybrids. Let the original game be H_0, and let q be an upper bound on the number of shareO queries by the adversary. We first define a hybrid $\hat{H}_0$ as follows. Let $\mathsf{ct}^* = (r^*, \mathsf{PRF.F}(k_s, r^*) + s_b)$ denote the encryption to s_b, i.e., the challenge secret, and let C^* denote the challenge access structure. The challenger aborts if for any query to the oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$, the challenger samples r^* when encrypting $s^{(i)}$. Since this event only occurs with probability $q/2^\lambda$, the advantage of the adversary in hybrid $\hat{H}_0$ only changes by this amount.

The rest of the proof follows the proof of tracer secrecy for iOGTSS. We define a hybrid H_1 as follows. We first sample a PRF keys k, k_s, then sample a bit b, and compute ct^* as the encryption of s i.e., $\mathsf{ct}^* \leftarrow\!\$ \mathsf{Enc}(k_s, (0||0||s))$. Let $\mathsf{ct}^* = (r^*, y^*)$. We then change how we sample DCS keys. Instead of using DCS.KeyGen to generate keys, the challenger uses DCS.ConstPK, with C_1 as defined in line 1 of the Share algorithm, and $C_2 : [n] \times \mathsf{CT}^2 \times \mathcal{C}_\lambda$ is defined as follows. For any input $(i, \mathsf{ct}, \mathsf{ct}' = (r, y), C')$, $C_2(i, \mathsf{ct}, \mathsf{ct}' = (r, y), C') = 1$ only if one of the following holds: (i) $r \neq r^*$, or (ii) $r = r^*$ and $\mathsf{Dec}(k, \mathsf{ct}) = (1, \cdot, \cdot)$, or (iii) if $r = r^*$, but $\mathsf{Dec}(k, \mathsf{ct}) = (0, \cdot, \cdot)$, then $y = y^*$ and $C = C$ and $i \in \mathcal{J}$. In other words, for the challenge public parameters (r^*, y^*, C^*), valid signatures exist only on ciphertexts that encrypt messages with first bit equal to one, for honest parties $i \in [n] \setminus \mathcal{J}$. If an adversary can distinguish between

$\underline{\mathsf{Setup}(1^\lambda)}$:

1. Sample PRF keys $k, k_s \leftarrow\!\!\$ \ \{0,1\}^\lambda$ and doubly-constrained signature keys $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_1) \leftarrow\!\!\$ \ \mathsf{DCS.KeyGen}(1^\lambda, \ell, z, C_1)$ where $C_1 : [n] \times \mathsf{CT}^2 \times \mathcal{C}_\lambda \to \{0,1\}$ is defined as, $C_1(i, \mathsf{ct}, \mathsf{ct}', C) = 1$ if and only if, $\mathsf{Dec}(k, \mathsf{ct}) = (1, \cdot, \cdot)$. / $\mathsf{CT} := \{0,1\}^\lambda \times \{0,1\}^{1+\log(n+1)+\log(|\mathcal{S}_\lambda|)}$ and $\ell = \ell(n, \lambda)$, $z = z(\lambda, n)$ will be specified later.
2. Obfuscate the program P' described below, i.e., $\hat{\mathsf{P}}' \leftarrow \mathsf{Obf}(\mathsf{P}')$.
3. Output $\mathsf{pp_r} := \hat{\mathsf{P}}'$ and $\mathsf{state} := (k, k_s, (\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_1))$.

$\underline{\mathsf{Share}(1^\lambda, s, C, \mathsf{state})}$:

1. Compute $\mathsf{ct}^* \leftarrow\!\!\$ \ \mathsf{Enc}(k_s, (0, 0, s))$.
2. For all $i \in [n]$, compute $\mathsf{ct}_i \leftarrow\!\!\$ \ \mathsf{Enc}(k, (0, 0, 0))$ and $\sigma_i := \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}_i, \mathsf{ct}^*, C))$. Set $\mathsf{sh}_i \leftarrow (i, \mathsf{ct}_i, \sigma_i)$.
3. Output $(\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp} \leftarrow (\mathsf{ct}^*, C), \mathsf{tk} \leftarrow (k, \mathsf{sk}_1), \mathsf{vk} \leftarrow \bot)$.

$\underline{\mathsf{P}'(\{i, \mathsf{ct}_i, \sigma_i\}_{i\in J}, \mathsf{pp})}$:

1. **Constants**: k, pk, k_s.
2. Parse pp as (ct^*, C).
3. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i, \mathsf{pp}), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
4. For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$. Compute $S_i := \{j_i + 1, \ldots, n\}$ and $S := \bigcap_{i\in\hat{J}} S_i$.
5. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
6. Set $\hat{J} = \hat{J} \cap S$. Abort if $C(x_{\hat{J}}) = 0$.
7. If $\forall\, i \in \hat{J}$, $f_i = 0$, then output s where $(\cdot, \cdot, s) \leftarrow \mathsf{Dec}(k_s, \mathsf{ct}^*)$.
8. Otherwise, output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

$\underline{\mathsf{Rec}(\mathsf{pp_r}, \mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J})}$

1. Parse $\mathsf{pp_r}$ as $\hat{\mathsf{P}}'$. Output $\hat{\mathsf{P}}'(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J})$.

Figure 16: Our iO-based scheme iOrGTSS with re-usable public parameters, that achieves $\emptyset$-strong traceability and tracer secrecy. The Enc, Dec and Trace algorithms are as defined in Figure 8

H_0 and H_1, it can be used to break security of the DCS scheme, as shown in the proof of Theorem 5.5. This is because, crucially, C_1 implies C_2, and the challenger does not use r^* when encrypting any other secret.

Next, we define a hybrid H_2, wherein we change the obfuscated program P' as follows. We hard-code $\mathcal{J}$ along with (ct^*, C^*) in P', and we abort if the input parameters pp equal (ct^*, C^*) and the input set J is a subset of $\mathcal{J}$, i.e., if $J \subseteq \mathcal{J}$. We argue that it is hard to distinguish between hybrids H_1 and H_2 because the functionality of the obfuscated program is the same in both. To see that, recall that $C^*(x_{\mathcal{J}}) = 0$, so, in hybrid H_1, if the program P' is run with input $J \subseteq \mathcal{J}$ and (ct^*, C^*), it would always be aborted in line 6, because $\hat{J} \subseteq J \subseteq \mathcal{J}$, which implies $C^*(x_{\hat{J}}) = 0$. Hence, it does not change functionality if we simply abort for such inputs.

We now make another hybrid H_3, wherein we change the program to abort when $\mathsf{pp} = (\mathsf{ct}^*, C^*)$, if for all non-corrupt input shares, i.e., $J \setminus \mathcal{J}$, the ciphertext decrypts to $(0, \cdot, \cdot)$ or the signature is not valid. Again, we argue that this does not change the functionality of the obfuscated program. To see that, observe that because the public key of DCS has been constrained, for parties $i \in J \setminus \mathcal{J}$ and for (ct^*, C^*), no valid signatures exist on ciphertexts which encrypt $(0, \cdot, \cdot)$. This means, that even in H_2, the program when given an input J such that for all these parties, the ciphertext encrypts $(0, \cdot, \cdot)$, we know that none of these signatures would verify, and hence they would not be included in $\hat{J}$. This implies that for such input sets J, the resulting $\hat{J}$ would end up being $\subseteq \mathcal{J}$ – which means $C^*(x_{\hat{J}}) = 0$, so such inputs would anyway be aborted in the second last line. Hence, adding the abort line in the beginning does not affect the functionality.

We define another hybrid H_4, wherein we puncture the key k_s at r^*, let $k_s^{(r^*)}$ denote the punctured key, and only store this key in the program instead of k_s. To see why this does not change functionality of the program, consider three types of inputs to the program: (i) $r \neq r^*$ where the input $\mathsf{pp} = (r, y, C)$. For such inputs, the program anyway does not need $\mathsf{PRF.F}(k_s, r^*)$, (ii) $r = r^*$, $(y, C) \neq (y^*, C^*)$. For such inputs, valid signatures exist only for i, ct_i such that ct_i decrypts to $(1, \cdot, \cdot)$. Hence, even if the set of input shares with valid signatures is authorized, the program will not query $\mathsf{PRF.F}(k_s, r^*)$ since it will output m_i where m_i comes from the decryptions of input shares. (iii) $r = r^*$ and $(y, C) = (y^*, C^*)$. For this case, the only valid inputs to the program which would ever reach the last line are those where $J \setminus \mathcal{J} \neq \emptyset$, and, for at least one party i^* in $J \setminus \mathcal{J}$, the signature is valid and the decryption $f_i = 1$. Hence, as per the logic of the program, i^* must be in $\hat{J}$. This means, that in the last line, the program will never output the decryption of ct^*, and just output m_{i^*}. In other words, the PRF is never evaluated at r^*. Hence, puncturing k_s at r^* does not alter the functionality.

Lastly, in hybrid H_5, we replace y^* with a uniformly random value. This is indistinguishable from H_4 by security of punctured PRF, since $\mathsf{PRF.F}(k_s, r^*)$ is indistinguishable from random for the adversary since its view only contains the punctured key $k_s^{(r^*)}$. Observe that in H_5, the view of the adversary contains no information about b, hence it can guess b in this game only with probability $1/2$. This proves the theorem. □

$\emptyset$-Strong Traceability. We prove $\emptyset$-Strong traceability in Theorem 5.14 below, by relying on the security of the signature scheme DCS, the obfuscation scheme Obf, and the puncturable PRF. Note that we focus on a less-adaptive notion of $\emptyset$-Strong traceability, wherein the adversary sends the label $\mathcal{I}$ along with the corrupt set $\mathcal{J}$ in the first step of the game.

Theorem 5.14. *If* DCS, Obf *and the puncturable* PRF *schemes are secure, then, for all* PPT *adversaries* $\mathcal{A}$,

$$\mathsf{Adv}^{\mathrm{sa-\emptyset-str-trac-re}}_{\mathcal{A},\mathsf{iOGTSS},\epsilon}(\lambda) = \mathsf{negl}(\lambda).$$

Proof. As in the proof of Theorem 5.6, we prove this theorem in two parts. In the first part, we prove that the tracer will blame at least one party if the reconstruction box R is minimally-useful. In the second part, we prove that the blamed party cannot be honest.

We start with the first part. Let us use s_0, s_1 to denote the secrets given by $\mathcal{A}$. Let $s = s_b$ denote the secret that is shared in the Share algorithm, and let $s^* = s_{b^*}$ denote the second secret that is sampled by the tracer, where both b and b^* are sampled uniformly at random from $\{0,1\}$. Let $\mathcal{I}$ denote the label of the reconstruction box R, and let $\mathcal{J}$ denote the set of corrupt parties.

We first prove that if the reconstruction box is minimally-useful, i.e., it outputs b with non-negligible probability when given honestly generated shares for the set $\mathcal{I}$, then it must output b^* with roughly the same probability, when given tracer-generated shares of the same set $\mathcal{I}$.

Lemma 5.15. *If the event* GoodBox *occurs (as defined in Fig. 15) and the* DCS, Obf *and the puncturable* PRF *schemes are secure, then,*

$$\Pr[R(\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}) = b^*] \geq \frac{1}{2} + \epsilon - \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,0}$ *are as defined in line 2(a)ii of the* Trace *algorithm.*

Proof. Let us use C^* to denote the challenge access structure. We first recall the definition of the event GoodBox: $\mathcal{I}\cap\mathcal{J} = \emptyset, C^*(x_{\mathcal{J}}) = 0, C^*(x_{\mathcal{I}}) = 0$ and R is a minimally-useful box w.r.t. $(C^*, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \mathsf{pp}_{\mathsf{r}}, \mathsf{pp}, s_0, s_1, b, \epsilon)$, where $\mathsf{sh}_i = (i, \mathsf{ct}_i, \sigma_i)$ are the honestly-generated shares of parties in $\mathcal{J}$.

By definition 4.4 and the fact that $\mathcal{J} \cap \mathcal{I} = \emptyset$, if we sample fresh shares of parties in $\mathcal{I}$ which are identically distributed as honest shares, then the box must output b with probability $\frac{1}{2} + \epsilon$. Formally,

$$\Pr[R(\{\mathsf{sh}'_i\}_{i\in\mathcal{I}}) = b] \geq \frac{1}{2} + \epsilon$$

where, for all $i \in \mathcal{I}$, sh'_i is sampled as follows: $\mathsf{ct}'_i \leftarrow\!\!\$\ \mathsf{Enc}(k, (0,0,0))$ and $\sigma'_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}'_i, \mathsf{pp}))$. Note that $\mathcal{J} \cap \mathcal{I}$ is crucial for this step, since if there was a party in $\mathcal{J} \cap \mathcal{I}$ then we cannot sample a fresh share independent of its original share sh_i.

Let us use H_0 to refer to this original game, wherein the challenger runs R with randomly sampled shares of $\mathcal{I}$ encrypting $(0,0,0)$, and the box outputs b with some probability. Let $\mathsf{ct}^* = (r^*, y^*)$. We will now define a series of hybrids:

- H'_0: Similar to $\hat{H}_0$ defined in the proof of Theorem 5.13, the challenger aborts if for any query to the oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$, the challenger samples r^* when encrypting $s^{(i)}$. Since this event only occurs with probability $q/2^\lambda$, the advantage of the adversary in hybrid H'_0 only changes by this amount.

- H'_1: The challenger samples the shares for $i \in \mathcal{J} \cup \mathcal{I}$ in the beginning, each containing a ciphertext encrypting $(0,0,0)$. Let us use (r_i, z_i) to denote the ciphertext in the shares for all $i \in \mathcal{J} \cup \mathcal{I}$. Then, for any query to the oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$, let $(r^{(i)}_j, z^{(i)}_j)$ denote the ciphertexts sampled by the challenger for shares of all $j \in [n]$. If there are any duplicates in the multi-set $\{r_i\}_{i\in\mathcal{J}\cup\mathcal{I}} \cup \{r^{(i)}_j\}_{i\in[q],j\in[n]}$, then the challenger aborts. Since this only happens with probability $((q+1)n)^2/2^\lambda$ which is negligible, this only affects the behavior of the box R by a negligible amount. The rest of the hybrids largely follow the proof of Theorem 5.7:

- H_1: Here, we change how we sample the keys of the doubly-constrained signature DCS. Specifically, the challenger first samples the ciphertexts for all parties in $\mathcal{J}\cup\mathcal{I}$, and the ciphertext encrypting the secret s_b i.e. $\mathsf{ct}^* = (r^*, y^*)$. Then, instead of running KeyGen as in the honest Share algorithm, the challenger samples a constrained key-pair, by running ConstPK. The input circuit C_1 is the same as that defined in line 1 of the Share algorithm, and the circuit C_2 is defined as follows: $C_2(i, \mathsf{ct}''_i = (r''_i, \hat{\mathsf{ct}}''_i), \mathsf{ct}'' = (r'', y''), C'')$ is 1 for the following inputs: (i) $r'' \neq r^*$ and $r''_i \notin \{r_j\}_{j\in\mathcal{J}\cup\mathcal{I}}$. (ii)

$r'' = r^*$ and $y'' = y^*, C'' = C^*$ and $i \in \mathcal{J} \cup \mathcal{I}$ and $\mathsf{ct}''_i = \mathsf{ct}_i$ for $i \in \mathcal{J}$ and $\mathsf{ct}''_i = \mathsf{ct}'_i$ for $i \in \mathcal{I}$, i.e., the ciphertext must match the ones sampled by the challenger for the challenge public parameters (ct^*, C^*). Note that C_1 is not a subset of C_2 as defined but that does not affect the proof since the adversary in this game never sees the tracing key.

We argue that if the box's behavior differs by a non-negligible amount from H'_1 to H_1, we can use it to break security of the DCS scheme. To see why, note that the view of the adversary only includes the obfuscated program which contains the public key pk, along with honestly generated shares of $\mathcal{J} \cup \mathcal{I}$ for (ct^*, C^*) and shares of $[n]$ for other secrets $(s^{(i)}, C^{(i)})$. Crucially, each of these shares contain a signature on messages that satisfy C_2 as defined above (because the abort conditions in H'_1 enforce that each $s^{(i)}$ is encrypted with randomness $r^{(i)}$ different from r^*, similarly the ciphertexts in all the shares are encrypted with different randomness). The reduction to DCS security is exactly as described in the proof of Theorem 5.7.

- H_2: Similar to H_2 in proof of Theorem 5.7, we now change the program P'. We hard-code $\mathsf{pp}^* = (\mathsf{ct}^*, C^*)$, the sets $\mathcal{J} \cup \mathcal{I}$ in the program, along with the shares $\{\mathsf{sh}_i = (i, (r_i, z_i), \sigma_i)\}_{i \in \mathcal{J}}$ and $\{\mathsf{sh}'_i = (i, (r_i, z_i), \sigma_i)\}_{i \in \mathcal{I}}$. Moreover, before checking any signatures, the program does the following. Let the input parameters $\mathsf{pp} = (r, y, C)$. (i) If $r = r^*$ but $\mathsf{pp} \neq \mathsf{pp}^*$, the program simply aborts. (ii) If $r = r^*$ and $\mathsf{pp} = \mathsf{pp}^*$, then we set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$ and for input shares of parties in $\mathcal{J} \cup \mathcal{I}$, we remove them from $\hat{J}$ if they do not match the hard-coded shares for these parties. (iii) If $r \neq r^*$, then, for each input share $(i, \mathsf{ct}_i = (r''_i, z''_i), \sigma_i)$, we remove i from $\hat{J}$ if $r''_i \in \{r_j\}_{j \in \mathcal{J} \cup \mathcal{I}}$. We argue that this hybrid is indistinguishable from H_1 by iO security since the change does not alter the functionality of the program. To see that, observe that there are no valid signatures for any party for $r = r^*$ but $\mathsf{pp} \neq \mathsf{pp}^*$, meaning that the resulting $\hat{J}$ would've been empty, meaning the program would've simply output $\perp$ for such inputs. Similarly, for any input share (i, ct_i) and $r \neq r^*$ such that $r''_i \in \{r_j\}_{j \in \mathcal{J} \cup \mathcal{I}}$, there is again no valid signature on such a share, meaning the program would've ignored share input anyway. Lastly for $\mathsf{pp} = \mathsf{pp}^*$, again observe that there are no valid signatures for parties outside $\mathcal{J} \cup \mathcal{I}$ so these parties would've been excluded from $\hat{J}$ anyway. Similarly, for parties in $\mathcal{J} \cup \mathcal{I}$, there exist valid signatures only for the shares hard-coded in the box. Hence, the resulting $\hat{J}$ is not affected by the change.

- H_3: We again change the program P'. Specifically, we puncture the PRF key k_s at r^*, we hard-code s_b in the program, and before the second-last line (which outputs s after checking all $f_i = 0$), we add another line which does the following: if $r = r^*$, then, it simply outputs $s = s_b$. We argue that this does not change the functionality of the program. Consider the case where $r \neq r^*$: in this case, the program anyway does not need to evaluate $\mathsf{PRF.F}(k_s, r^*)$, and the added line and the puncturing makes no difference. In the case where $r = r^*$, note that we only need to worry about the case where $\mathsf{pp} = \mathsf{pp}^*$ since the program aborts anyway if this is not the case. For such inputs, observe that the resulting $\hat{J}$ (before the execution of the new line) is an authorized subset of $\mathcal{J} \cup \mathcal{I}$, and the corresponding shares are the hard-coded shares each of which encrypt $(0, \cdot, \cdot)$. This means, for such an input, the program would always output $\mathsf{Dec}(k_s, \mathsf{CT}^*) = \mathsf{PRF.F}(k_s, r^*) + y^*$ which is equal to $0||0||s_b$ by construction. Hence, the modified program in this hybrid is functionally equivalent to the one in H_2.

- H_4: We again modify the obfuscated program: For the case $r = r^*$ and $\mathsf{pp} = \mathsf{pp}^*$, the program computes $J \cap (\mathcal{J} \cup \mathcal{I})$, and only keeps parties for which the shares match the hard-coded shares (as before), but now, the program does not decrypt the ciphertexts for the remaining shares for this case. Figure 17 depicts this program. Essentially, we can think of this program as two sub-programs: one for $r = r^*$ and $\mathsf{pp} = \mathsf{pp}^*$, and another for $r \neq r^*$.

$\underline{\mathsf{P}'(\mathsf{pp}, \{i, \mathsf{ct}_i, \sigma_i\}_{i\in J})}$:

1. **Constants:** $k, k_s^{(r^*)}, \mathsf{pk}, \mathsf{pp}^* = (r^*, y^*, C^*)$, and $\mathcal{J}, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, \{\mathsf{sh}'_i\}_{i\in\mathcal{I}}$ and s_b.
2. Parse pp as (r, y, C) and $\mathsf{ct}_i = (r''_i, z''_i)\ \forall\ i \in J$. If $r = r^*$ but $(y, C) \neq (y^*, C^*)$, output $\perp$.
3. If $r = r^*$:
   (a) Set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$.
   (b) For all $i \in J \cap \mathcal{J}$, if (i, ct_i) does not match sh_i then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
   (c) For all $i \in J \cap \mathcal{I}$, if (i, ct_i) does not match sh'_i then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
   (d) For all $i \in \mathcal{J}$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i, \mathsf{pp}), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
4. If $r \neq r^*$, let $\hat{J} = J$.
   (a) For all $i \in J$, if $r''_i \in \{r_i\}_{\mathcal{J}\cup\mathcal{I}}$, then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
   (b) For any $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i, \mathsf{pp}), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
   (c) For all $i \in \hat{J}$, let $(f_i, j_i, m_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
   (d) For all $i \in \hat{J}$, compute $S_i = \{j_i + 1, \ldots, n\}$, and let $S = \cap_{i\in\hat{J}} S_i$.
   (e) For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
   (f) Set $\hat{J} \leftarrow \hat{J} \cap S$.
5. Abort if $C(x_{\hat{J}}) = 0$.
6. If $r = r^*$, output s_b.
7. Else, if $\forall i \in \hat{J}$, $f_i = 0$, output s where $(\cdot||\cdot||s) = \mathsf{Dec}(k_s^{(r^*)}, \mathsf{ct})$.
8. Otherwise, output m_{i^*} where i^* is the smallest index s.t. $f_i = 1$, i.e. $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

Figure 17: The modified program used in hybrid H_4 in the $\emptyset$-strong traceability proof for our scheme iOrGTSS with re-usable public parameters.

We argue that the functionality of the program is not affected by this change. Observe that for $\mathsf{pp} = \mathsf{pp}^*$, for any input set J, $\hat{J}$ is set to $J \cap (\mathcal{J} \cup \mathcal{I})$, and even for the remaining parties in J, the program keeps the shares that match the hard-coded shares. Hence, the resulting set $\hat{J}$ will be a subset of $\mathcal{J} \cup \mathcal{I}$, with each share containing an encryption to $(0, 0, 0)$ along with valid signatures. This means that the resulting $\hat{J}$ would indeed remain the same, even if we did the decryption and computed $\hat{J} \cap S$, since S would simply be $[n]$.

- H_5: We now puncture the PRF key k. Let $\mathsf{sh}_i = (i, (r_i, ...))$ for all $i \in \mathcal{J}$, and $\mathsf{sh}'_i = (i, (r_i,))$ for all $i \in \mathcal{I}$. We puncture k at the set $T := \{r_i\}_{i\in\mathcal{I}\cup\mathcal{J}}$. Note that we do not need to hard-code $z_r := \mathsf{PRF.F}(k, r)$ for $r \in T$ in the program since it never needs these evaluations. This is because, in the case $r = r^*$, the program does not need to decrypt the shares, and for $r \neq r^*$, the program anyway ignores shares containing any value in T. Hence, H_5 is computationally indistinguishable from H_4 to $\mathcal{A}$ by security of obfuscation.

- H_6: We can now replace the ciphertext in the hard-coded shares for all $i \in \mathcal{J} \cup \mathcal{I}$ with a uniformly random value. This is indistinguishable from H_5 by security of punctured PRFs.

We will now consider a different series of hybrids, starting from $\hat{H}_0$, wherein we run the box R with shares generated by the tracer, i.e., $\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}$. Here, each $\mathsf{sh}^*_{i,0}$ contains a ciphertext $\mathsf{ct}^*_{i,0}$ that encrypts $(1, 0, s_{b^*})$ and a signature $\sigma^*_{i,0}$ on $(i, \mathsf{ct}^*_{i,0}, \mathsf{pp}^*)$ generated using the constrained key sk_1. Let us use p^* to denote the

probability that R outputs b^* given these inputs. Also, let us use (r_i, z_i) to denote the ciphertexts in shares of $\mathcal{J} \cup \mathcal{I}$ (in the tracer-generated shares for $\mathcal{I}$). We now define more hybrids, starting from $\hat{H}_0$:

- $\hat{H}_1$: Similar to $\hat{H}_1$ in the proof of Theorem 5.7, we change the signatures $\sigma^*_{i,0}$ to be generated by the unconstrained signing key sk instead of sk_1. Since these signatures are on messages that satisfy the constraint C_1, this hybrid is computationally indistinguishable from $\hat{H}_0$ by the identical-distribution property of DCS.

- $\hat{H}'_0$: Similar to H'_0 defined above, the challenger aborts if for any query to the oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$, the challenger samples r^* when encrypting $s^{(i)}$. Since this event only occurs with probability $q/2^\lambda$, the advantage of the adversary in hybrid H'_0 only changes by this amount.

- $\hat{H}'_1$: Similar to H'_1 defined above, the challenger samples the shares for $i \in \mathcal{J} \cup \mathcal{I}$ in the beginning, each containing a ciphertext encrypting $(0, 0, 0)$. Let us use (r_i, z_i) to denote the ciphertext in the shares for all $i \in \mathcal{J} \cup \mathcal{I}$. Then, for any query to the oracle $\mathsf{shareO}(s^{(i)}, C^{(i)})$, let $(r^{(i)}_j, z^{(i)}_j)$ denote the ciphertexts sampled by the challenger for shares of all $j \in [n]$. If there are any duplicates in the multi-set $\{r_i\}_{i\in\mathcal{J}\cup\mathcal{I}} \cup \{r^{(i)}_j\}_{i\in[q],j\in[n]}$, then the challenger aborts. Since this only happens with probability $((q+1)n)^2/2^\lambda$ which is negligible, this only affects the behavior of the box R by a negligible amount. The rest of the hybrids largely follow the proof of Theorem 5.7:

- $\hat{H}_2$: Similar to H_1 above, we generate a constrained key-pair for the DCS. We run $\mathsf{ConstPK}$ with C_1 as in the real scheme, and the circuit C_2 is defined as follows: $C_2(i, \mathsf{ct}''_i = (r''_i, \hat{\mathsf{ct}}''_i), \mathsf{ct}'' = (r'', y''), C'')$ is 1 for the following inputs: (i) $r'' \neq r^*$ and $r''_i \notin \{r_j\}_{j\in\mathcal{J}\cup\mathcal{I}}$. (ii) $r'' = r^*$ and $y'' = y^*, C'' = C^*$ and $i \in \mathcal{J} \cup \mathcal{I}$ and $\mathsf{ct}''_i = \mathsf{ct}_i$ for $i \in \mathcal{J}$ and $\mathsf{ct}''_i = \mathsf{ct}^*_{i,0}$ for $i \in \mathcal{I}$, i.e., the ciphertext must match the ones sampled by the challenger for the challenge public parameters (ct^*, C^*). Note that C_1 is not a subset of C_2 as defined but that does not affect the proof since the adversary in this game never sees the tracing key. By an argument similar to H_1 above, this hybrid is indistinguishable from $\hat{H}'_1$.

- $\hat{H}_3$: Similar to H_2 above, we hard-code the set $\mathcal{J} \cup \mathcal{I}$ and the corresponding shares in the program. Then, if $r = r^*$ but $\mathsf{pp} \neq \mathsf{pp}^*$, the program simply aborts. If $r = r^*$ and $\mathsf{pp} = \mathsf{pp}^*$, the program only keeps shares that are subset of $\mathcal{J} \cup \mathcal{I}$ such that the input shares match the hard-coded shares. For $r \neq r^*$, it removes any share whose encryption uses any value in $\{r_i\}_{i\in\mathcal{J}\cup\mathcal{I}}$.

- $\hat{H}_4$: Similar to H_3 above, we change the program P'; we puncture k_s at r^* and hard-code s_{b^*} into the program, and add a line that simply outputs s_{b^*} if $r = r^*$ and the resulting set $\hat{J}$ is authorized. We argue that this change does not alter the functionality of the program. As argued for H_3 above, for $r \neq r^*$, these changes do not make a difference. For $r = r^*$, we only need to worry about the case $\mathsf{pp} = \mathsf{pp}^*$, in which case, the resulting $\hat{J}$ is a subset of $\mathcal{J} \cup \mathcal{I}$, and the corresponding shares must be the hard-coded shares. Next, if the resulting $\hat{J}$ is a subset of $\mathcal{J}$, then the program will abort in both hybrids since $C^*(x_\mathcal{J}) = 0$. On the other hand, if $\hat{J}$ has at least one share from $\mathcal{I}$, say $\mathsf{sh}^*_{i,0}$, observe that this share decrypts to $(1, \cdot, s_{b^*})$, which means that the program will output s_{b^*} for any such input if $\hat{J}$ is authorized. Hence, $\hat{H}_4$ is computationally indistinguishable from $\hat{H}_3$ by iO security.

- $\hat{H}_5$: Similar to H_4, we now remove the logic in the program that decrypts the ciphertexts in the input shares for the case $r = r^*$. By a similar argument as in H_4, this does not alter the functionality of the program.

- $\hat{H}_6$: Similar to H_5, we now puncture the key k at the set T as defined above. Again, we do not hard-code the PRF evaluations at T since the program does not need them. By a similar argument as H_5, this is indistinguishable from $\hat{H}_5$ by iO security.

- $\hat{H}_7$: We now replace the ciphertexts in the shares of $\mathcal{J} \cup \mathcal{I}$ with uniformly random values as in H_6 above. This is computationally indistinguishable from $\hat{H}_6$ by security of punctured PRFs.

Now, we argue that the hybrids H_6 and $\hat{H}_7$ are indistinguishable: the key difference in the two hybrids is the hard-coded secret is s_b and s_{b^*} respectively, but they both come from the same distribution. Moreover, all the hard-coded ciphertexts and shares are identically distributed in the two hybrids (and do not contain any information about b or b^*). Hence, if R outputs b in H_6 with some probability, it must output b^* with almost the same probability in $\hat{H}_7$. □

The following lemma proves that the box cannot output b^* with probability much more than $1/2$ when given shares with ciphertexts encrypting $(1, n, s^*)$.

Lemma 5.16. *In the event* GoodBox*,*

$$\Pr\left[R\left(\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}\right) = b^*\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,n}$ *shares are as defined in the* Trace *algorithm.*

Proof. The proof follows essentially from the proof of Theorem 5.8. We define a series of hybrids: H_0 denotes the original game, then $\hat{H}_0$ is defined with signatures in $\mathsf{sh}^*_{i,n}$ being generated using an unconstrained signing key.

We first define hybrids H'_0 and H'_1 as in the proof of Theorem 5.15, wherein the challenger aborts if it uses r^* when encrypting a secret for any shareO query, or if it re-uses the randomness for ciphertexts in any two shares that it sampled.

In H_1, we sample a constrained key-pair for DCS, wherein C_1 and C_2 are constructed exactly as in hybrid H_1 in proof of Theorem 5.15.

Next, in H_2, we change the program similar to H_2 in proof of Theorem 5.15: we hard-code the sets $\mathcal{J}, \mathcal{I}$ and their shares.

In hybrid H_3, we modify the program again. We add a line before the last two lines, which simply outputs $\perp$ if $r = r^*$, and we puncture k_s at r^*. We argue that this does not change the functionality. For inputs with $r \neq r^*$, this change does not affect the behavior. Then, for $r = r^*$ and $\mathsf{pp} = \mathsf{pp}^*$, observe that the resulting set $\hat{J}$ is always a subset of $\mathcal{J} \cup \mathcal{I}$. Moreover, if $\hat{J}$ is a subset of $\mathcal{J}$, then the program would abort anyway since $C^*(x_{\mathcal{J}}) = 0$. So we only need to worry about inputs such that the resulting $\hat{J}$ has at least one share from $\mathcal{I}$. But, the share for any party in $\mathcal{I}$ encrypts $(1, n, s_{b^*})$. This means that the set S will end up being a subset of $\mathcal{I}$, and as a result, $\hat{J} \subseteq \mathcal{I}$. But since $C^*(x_{\mathcal{I}}) = 0$, the program will abort for these inputs as well. Hence, it ends up always aborting if $r = r^*$.

In hybrid H_4, similar to H_4 in proof of Theorem 5.15, we do not run Dec on the ciphertexts in shares of $\mathcal{J} \cup \mathcal{I}$ if $r = r^*$. By a similar argument, this does not affect the functionality of the program.

In hybrid H_5, we puncture the PRF key k at the set T (as defined in hybrid H_5 in proof of Theorem 5.15). Observe that this does not change functionality since the program does not need to evaluate $\mathsf{PRF.F}(k, \cdot)$ at inputs in T anyway.

In hybrid H_6, we modify the ciphertexts for all $\mathcal{J} \cup \mathcal{I}$ to be uniformly random. This is indistinguishable from H_5 by security of punctured PRFs.

But now observe that in hybrid H_6, the view of the adversary contains no information about b^*, hence the box R cannot output b^* with probability more than $1/2$. □

Similar to the proof of Theorem 5.6, the above two lemmas imply that the Trace algorithm will blame at least one party $j^* \in [n]$ if the event GoodBox occurs. We now move to the second part, which proves that the jump cannot be honest. Crucially, this holds even if the adversary corrupts upto $n-1$ parties, and chooses arbitrary $\mathcal{I}, \mathcal{J}$, and the box R may not be minimally-useful. Following the proof of Theorem 5.6, we prove this via two lemmas.

Lemma 5.17. *Let $\hat{p}_j$ denote the probability with which R outputs b^* when run with shares $\{\mathsf{sh}^*_{i,j}\}_{i\in\mathcal{I}}$. For all $j \in [n] \setminus (\mathcal{J} \cup \mathcal{I})$,*

$$|\hat{p}_j - \hat{p}_{j-1}| \leq \mathsf{negl}(\lambda).$$

Proof. The proof follows the proof of Theorem 5.9 but with two additional hybrids H'_0, H'_1 as in the proof of Theorem 5.15. Let H_0 denote the original game where R is run with shares encrypting $(1, j-1, s_{b^*})$.

We first define hybrids H'_0 and H'_1 as in the proof of Theorem 5.15, wherein the challenger aborts if it uses r^* when encrypting a secret for any shareO query, or if it re-uses the randomness for ciphertexts in any two shares that it sampled.

We define $\hat{H}_0$, wherein the signatures in the shares $\mathsf{sh}^*_{i,j-1}$ are generated using the unconstrained signing key.

We then define H_1, wherein the challenger samples a constrained key-pair, with C_1 as in the scheme, and $C_2(i, \mathsf{ct}''_i = (r''_i, z''_i), (\mathsf{ct}'' = (r'', y''), C''))$ defined as follows: (i) $r'' \neq r^*$ and $r''_i \notin \{r_j\}_{j\in\mathcal{J}\cup\mathcal{I}}$ where r_j is the randomness used in ciphertexts in shares of $\mathcal{J} \cup \mathcal{I}$. (ii) $r'' = r^*$ and $y'' = y^*, C'' = C^*$, and the same constraints as in C_2 defined in hybrid H_1 in proof of Theorem 5.9, i.e. either $i \neq j$ or $i \in \mathcal{J} \cup \mathcal{I}$ and ct''_i matches the ciphertext sampled by the challenger for i. In other words, there are no valid signatures for party j with respect to pp^*. By a similar argument as in H_1 hybrid of Theorem 5.9, this hybrid is indistinguishable from $\hat{H}_0$ by security of DCS.

Next, we define H_2 where we change the program P'. This is a combination of H_2 in proof of Theorem 5.9 and H_2 in proof of Theorem 5.15. We hard-code pp^*, the set $\mathcal{J} \cup \mathcal{I}$ and their corresponding shares, and the index j, in the program. Then, the program does the following: (i) If $r = r^*$ but $\mathsf{pp} \neq \mathsf{pp}^*$, the program simply aborts. (ii) If $r = r^*$ and $\mathsf{pp} = \mathsf{pp}^*$, then we set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I} \cup \{j\})$ and for input shares of parties in $\mathcal{J} \cup \mathcal{I}$, we remove them from $\hat{J}$ if they do not match the hard-coded shares for these parties. (iii) If $r \neq r^*$, then, for each input share $(i, \mathsf{ct}_i = (r''_i, z''_i), \sigma_i)$, we remove i from $\hat{J}$ if $r''_i \in \{r_j\}_{j\in\mathcal{J}\cup\mathcal{I}}$. This does not change the functionality of the program. For cases (i) and (iii), the argument from H_2 in proof of Theorem 5.15 applies directly. For case (ii), we use the same argument as H_2 in proof of Theorem 5.9: since there is no valid signature for j for pp^*, this would anyway be removed from $\hat{J}$. Similarly for parties in $\mathcal{J} \cup \mathcal{I}$, valid signatures exist only on the ciphertexts in the hard-coded shares, so they would also be removed if the input shares did not match.

The rest of the hybrids H_3, H_4 are defined exactly as in the proof of Theorem 5.9. By combining arguments from that proof and the proof of Theorem 5.15, we prove the lemma. □

To finish the proof, in the lemma below, we prove that the tracer will never blame any honest party in $\mathcal{I}$.

Lemma 5.18. *For all $j \in \mathcal{I} \setminus \mathcal{J}$,*

$$|\hat{p}_j - \hat{p}_{j-1}| \leq \mathsf{negl}(\lambda).$$

Proof. We use a series of hybrids, by following the proof of Theorem 5.10, with modifications for the reusability based on proof of Theorem 5.15.

Let H_0 be the hybrid where the box is run with inputs $\mathsf{sh}^*_{i,j-1}$ for $i \in \mathcal{I}$. We first define hybrids H'_0 and H'_1 as in the proof of Theorem 5.15, wherein the challenger aborts if it uses r^* when encrypting a secret for any shareO query, or if it re-uses the randomness for ciphertexts in any two shares that it sampled.

We then define $\hat{H}_0$, where the signatures in $\{\mathsf{sh}^*_{i,j-1}\}$ are generated using the unconstrained signing key.

In hybrid H_1, we constrain the keys for DCS, such that C_2 is one for either case (i) as defined in H_1 in proof of Theorem 5.17, and for (ii) $r'' = r^*$ and $y'' = y^*, C'' = C^*$, and the same constraints as in C_2 defined in hybrid H_1 in proof of Theorem 5.10, i.e. either $i \notin (\mathcal{I} \cup \mathcal{J})$ or if $i \in \mathcal{I} \cup \mathcal{J}$, the ciphertext matches the one sampled by the challenger for this i. In particular, for j, the only valid ciphertext is the

one sampled by the challenger which encrypts $(1, j-1, s^*)$ (since $j \notin \mathcal{J}$). It follows from the proofs of Theorems 5.10 and 5.15 that this hybrid is computationally indistinguishable from $\hat{H}_0$ by DCS security.

We then define hybrid H_2 exactly as the H_2 defined in the proof of Theorem 5.15, we hard-code pp^*, along with sets $\mathcal{J}, \mathcal{I}$ and their corresponding shares.

The rest of the hybrids are exactly as defined in the proof of Theorem 5.9. □

Combining all the above lemmas as in the proof of Theorem 5.6, we complete the proof. □

5.4 Achieving Tracer Non-imputability

Figures 18, 19 and 20 formally present the scheme iOGTSSNI that achieves tracer non-imputability, with changes from the previous construction highlighted in blue. Similar to before, we use a PRF $\mathsf{PRF}' : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\} \times \{0,1,\ldots,n\} \times \{0,1,\ldots,\lambda\} \times \{0,1\}$, a doubly-constrained signature scheme DCS, indistinguishability obfuscation Obf and a one-way function F.

$\underline{\mathsf{Share}(1^\lambda, s, C)}$:

1. Sample PRF key $k \leftarrow\!\!\$\ \{0,1\}^\lambda$ and doubly-constrained signature keys $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_1) \leftarrow\!\!\$\ \mathsf{DCS.KeyGen}(1^\lambda, \ell, z, C_1)$ where $C_1 : [n] \times \mathsf{CT} \to \{0,1\}$ is defined as, $C_1(i, \mathsf{ct}) = 1$ if and only if, $\mathsf{Dec}(k, \mathsf{ct}) = (1, \cdot)$. / $\ell = \log(n) + 3\lambda + 2 + \log(n+1)$, $\mathsf{CT} := \{0,1\}^\lambda \times \{0,1\}^{2+\lambda+\log(n+1)+\log(|\mathcal{S}_\lambda|)}$.
2. For all $i \in [n]$, sample $x_i \leftarrow\!\!\$\ \{0,1\}^\lambda$, compute $\mathsf{ct}_i := \mathsf{Enc}(k, (0,0,0,0,0))$ and $\sigma_i := \mathsf{DCS.Sign}(\mathsf{sk}, (i, x_i, \mathsf{ct}_i))$. Set $\mathsf{sh}_i \leftarrow (i, x_i, \mathsf{ct}_i, \sigma_i)$.
3. Set $\mathsf{tk} \leftarrow (k, \mathsf{sk}_1, \{F(x_1), \ldots, F(x_n)\})$, $\mathsf{vk} \leftarrow \{F(x_1), \ldots, F(x_n)\}$.
4. Obfuscate the program P described below, i.e., $\hat{\mathsf{P}} \leftarrow \mathsf{Obf}(\mathsf{P})$.
5. Output $\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp} \leftarrow \hat{\mathsf{P}}, \mathsf{tk}, \mathsf{vk}$.

$\underline{\mathsf{Enc}(k, (f, j, m, u, b))}$:

1. Sample $r \leftarrow\!\!\$\ \{0,1\}^\lambda$.
2. Compute $\mathsf{ct}' \leftarrow \mathsf{PRF}'.\mathsf{F}(k, r) + (f||j||m||u||b)$.
3. Output $\mathsf{ct} \leftarrow (r, \mathsf{ct}') \in \mathsf{CT}$.

$\underline{\mathsf{Dec}(k, \mathsf{ct})}$:

1. Parse ct as (r, ct').
2. Output $\mathsf{ct}' - \mathsf{PRF.F}(k, r)$.

$\underline{\mathsf{Rec}(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J})}$:

1. Parse pp as $\hat{\mathsf{P}}$.
2. Output $\hat{\mathsf{P}}(\{\mathsf{sh}_i\}_{i\in J})$.

Figure 18: The Share and Rec algorithms for our iO-based Traceable secret sharing scheme iOGTSSNI, with tracer non-imputability. The algorithm P is described in Fig. 19.

Correctness. Correctness follows from correctness of the doubly constrained signature and obfuscation. Specifically, for any set $J \subseteq [n]$ such that $C(x_J) = 1$, when the obfuscated program is run with $\{\mathsf{sh}_i = (i, x_i, \mathsf{ct}_i, \sigma_i)\}_{i\in J}$ as input, all the signatures will verify successfully, by correctness of DCS. Moreover, since each ciphertext ct_i decrypts to $(0,0,0,0)$, we will get $S = [n]$. Hence, $\hat{J} = J$ and the program will not abort in line 7, and since $f_i = 0$ for all input shares, the program will indeed output the correct secret s.

Tracer secrecy. Theorem 5.19 below proves tracer secrecy of our scheme based on the security of the DCS and Obf schemes.

$\mathsf{P}(\{i, x_i, \mathsf{ct}_i, \sigma_i\}_{i\in J})$:

1. **Constants**: C, k, pk, s.
2. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, x_i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
3. For all $i \in \hat{J}$, let $(f_i, j_i, m_i, u_i, b_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
4. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1 + f_i, \ldots, n\}$. If $f_i = 1$: if $u_i = 0$ or $((j_i + 1) \in \hat{J}$ and $x_{j_i+1}[u_i] = 1 - b_i)$, then set $S_i \leftarrow S_i \cup \{j_i + 1\}$.
5. Compute $S \leftarrow \bigcap_{i\in\hat{J}} S_i$. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
6. Set $\hat{J} = \hat{J} \cap S$.
7. Abort if $C(x_{\hat{J}}) = 0$.
8. If $\forall\, i \in \hat{J}$, $f_i = 0$, then output s.
9. Otherwise, output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

Figure 19: The algorithm P for our iO-based Traceable secret sharing scheme iOGTSSNI, with tracer non-imputability.

$\mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I})$:

1. Parse tk as $(k, \mathsf{sk}_1, \{y_i := F(x_i)\}_{i\in[n]})$. Sample $b^* \leftarrow\!\$\ \{0,1\}$ and set $s^* := s_{b^*}$.
2. Let $\mathcal{J}^* := \perp$. For $j \in \{0, \ldots, n\}$:
   (a) Let $p_j \leftarrow 0$. For $q \in \{1, \ldots, N\}$: // $N = \frac{8\lambda\cdot n^2}{\epsilon^2}$
      i. For all $i \in \mathcal{I}$, sample $x'_i \leftarrow\!\$\ \{0,1\}^\lambda$, compute $\mathsf{ct}'_i \leftarrow \mathsf{Enc}(k, (1, j, s^*, 0, 0))$ and $\sigma'_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}_1, (i, x'_i, \mathsf{ct}'_i))$.
      ii. Query R on $\{\mathsf{sh}^*_{i,j} \leftarrow (i, x'_i, \mathsf{ct}'_i, \sigma'_i)\}_{i\in\mathcal{I}}$. Let b denote R's response.
      iii. If $s_b = s^*$, then set $p_j \leftarrow p_j + 1$.
   (b) Compute $p'_j \leftarrow p_j/N$. If $j > 0$ and $p'_{j-1} - p'_j > \epsilon/4n$, then set $\mathcal{J}^* \leftarrow \mathcal{J}^* \cup \{j\}$.
3. Let $\pi \leftarrow \perp$. For each $j \in \mathcal{J}^*$:
   (a) Let $x^*_j \leftarrow 0^\lambda$. For $u \in [\lambda]$:
      i. Let $c_{j-1,u} := 0$.
      ii. For $q \in \{1, \ldots, N\}$:
         A. For all $i \in \mathcal{I}$, sample $x''_i \leftarrow\!\$\ \{0,1\}^\lambda$, compute $\mathsf{ct}''_i \leftarrow \mathsf{Enc}(k, (1, j-1, s^*, u, 0))$ and $\sigma''_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}_1, (i, x''_i, \mathsf{ct}''_i))$.
         B. Query R on $\{\mathsf{sh}^*_{i,j-1,u} := (i, x''_i, \mathsf{ct}''_i, \sigma''_i)\}_{i\in\mathcal{I}}$. Let b denote R's response.
         C. If $s_b = s^*$ then set $c_{j-1,u} \leftarrow c_{j-1,u} + 1$.
      iii. Compute $c'_{j-1,u} := c_{j-1,u}/N$. If $|c'_{j-1,u} - p'_{j-1}| \geq \epsilon/4n$ then set $x^*_j[u] \leftarrow 0$ else $x^*_j[u] \leftarrow 1$.
   (b) If $F(x^*_j) = y_j$ then set $\pi \leftarrow \pi \cup \{(j, x^*_j)\}$. Otherwise, set $\mathcal{J}^* \leftarrow \mathcal{J}^* \setminus \{j\}$.
4. Output $\mathcal{J}^*$ and π.

$\mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi)$:

1. Parse π as $\{(j, x^*_j)\}_{j\in\mathcal{J}^*}$, and vk as $\{y_1, \ldots, y_n\}$.
2. Output 1 if for all $j \in \mathcal{J}^*$, $F(x^*_j) = y_j$.

Figure 20: The Trace and Verify algorithms for our iO-based Traceable secret sharing scheme iOGTSSNI, with tracer non-imputability.

Theorem 5.19. *For all* PPT *adversaries D, there exist adversaries, such that,*

$$\mathsf{Adv}^{\mathrm{t-sec}}_{D,\mathsf{iOGTSS}}(\lambda) \leq \mathsf{Adv}^{\mathrm{sec}}_{D_1,\mathsf{DCS}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_2,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_3,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{iosec}}_{D_4,\mathsf{Obf}}(\lambda)$$

Proof of Thm. 5.19. The proof is the same as the proof of Theorem 5.5. We define a series of hybrids to constrain signatures for honest parties to be valid only for ciphertexts encrypting $(1, \cdot, \cdot)$. This eventually allows us to remove the actual secret s from the program P. Hence, the advantage of any PPT adversary is bounded by $1/2 + \mathsf{negl}(\lambda)$. This proves the theorem. □

$\emptyset$-Strong Traceability. Theorem 5.20 below proves $\emptyset$-Strong traceability of our scheme, by relying on the security of DCS, puncturable PRFs and obfuscation.

Theorem 5.20. *For all* PPT *adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\emptyset-\mathrm{str-trac}}_{\mathcal{A},\mathsf{iOGTSSNI},\epsilon}(\lambda) \leq \mathsf{negl}(\lambda).$$

Proof of Thm. 5.20. Similar to proof of Thm. 5.6, we prove this theorem in two parts. In the first part, we prove that the tracer will blame at least one party if the reconstruction box R is minimally-useful. In the second part, we prove that the blamed party cannot be honest.

We start with the first part. Let us use s_0, s_1 to denote the secrets given by $\mathcal{A}$. Let $s = s_b$ denote the secret that is shared in the Share algorithm, and let $s^* = s_{b^*}$ denote the second secret that is sampled by the tracer, where both b and b^* are sampled uniformly at random from $\{0, 1\}$. Let $\mathcal{I}$ denote the label of the reconstruction box R, and let $\mathcal{J}$ denote the set of corrupt parties.

Similar to Lemma 5.7, we first prove that if the reconstruction box is minimally-useful, i.e., it outputs b with non-negligible probability when given honestly generated shares for the set $\mathcal{I}$, then it must output b^* with roughly the same probability, when given tracer-generated shares of the same set $\mathcal{I}$.

Lemma 5.21. *If the event* GoodBox *occurs (as defined in Fig. 3) and the* DCS, Obf *and the puncturable* PRF *schemes are secure, then,*

$$\Pr[R(\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}) = b^*] \geq \frac{1}{2} + \epsilon - \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,0}$ are as defined in line 2(a)ii of the Trace *algorithm in Fig. 20.*

Proof. The proof is essentially the same as that for Lemma 5.7. The only difference is that ciphertexts now encrypt $(0, 0, 0, 0, 0)$ in hybrid H_0, and $(1, 0, s^*, 0, 0)$ in $\hat{H}_0$, and all shares sh_i include a randomly sampled identifier x_i. □

Next, Lemma 5.22 below proves that the box cannot output b^* with probability more than $1/2$ when given shares with ciphertexts encrypting $(1, n, 0, 0)$.

Lemma 5.22.

$$\Pr[R(\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}) = b^*] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,n}$ shares are as defined in line 2(a)ii of the Trace *algorithm in Fig. 20.*

Proof. The proof is the same as that for Lemma 5.8. □

Let $\hat{p}_j$ denote the probability with which the box R outputs b^* when run with shares $\{\mathsf{sh}^*_{i,j}\}_{i\in\mathcal{I}}$. By analysis similar to that in proof of Thm. 5.6, there must exist at least one $j^* \in [n]$ such that $\hat{p}_{j^*-1} - \hat{p}_{j^*} > \epsilon/2n$. Moreover, $p'_{j^*-1} - p'_{j^*} > \epsilon/4n$ except with negligible probability. This proves that the tracer will find at least one party i.e., j^* at the end of step 2 of the tracing algorithm.

We now prove that the tracer will indeed find the correct identifier x_{j^*} for this party. We introduce some notation first. For any $i \in \mathcal{J}$, let $\mathsf{sh}_i = (i, x_i, \mathsf{ct}_i, \sigma_i)$. For any $j \in [n]$, let $\hat{c}_{j,u,0}$ denote the probability with which the box R outputs b^* when run with shares $\{\mathsf{sh}^*_{i,j,u}\}_{i\in\mathcal{I}}$, as defined in line 3(a)iiB of the Trace algorithm in Fig. 20. We have, $\hat{c}_{j,0,0} = \hat{p}_j$ by definition. We now prove two lemmas:

Lemma 5.23. *In the event* GoodBox*, if $j^* \in \mathcal{J}$, for all $u \in [\lambda]$,*

$$|\hat{c}_{j^*-1,0,0} - \hat{c}_{j^*-1,u,0}| \leq \mathsf{negl}(\lambda),$$

if $x_{j^*}[u] = 1$.

Proof. We prove this with a series of hybrids. Let H_0 denote the game where the challenger runs the Share algorithm honestly, and runs R with shares $\{\mathsf{sh}^*_{i,j^*-1}\}_{i\in\mathcal{I}}$ as defined in the Trace algorithm.

We start with a hybrid $\hat{H}_0$, wherein we change how the shares sh^*_{i,j^*-1} are sampled. Specifically, the challenger generates the signature in these shares using the unconstrained secret key sk instead of sk_1. Because these signatures are on messages that satisfy the constraint C_1, by the identical-distribution property of DCS, no PPT adversary can distinguish between hybrids H_0 and $\hat{H}_0$.

We now define hybrid H_1, wherein the challenger first samples identifiers and ciphertexts for all parties in $\mathcal{J} \cup \mathcal{I}$. Specifically, it samples the PRF key k, then $x_i \leftarrow\!\!\$\ \{0,1\}^\lambda$, $\mathsf{ct}_i \leftarrow \mathsf{Enc}(k, (0,0,0,0,0))$ for all parties i in $\mathcal{J}$ conditioned on $x_{j^*}[u] = 1$. For all parties $i \in \mathcal{I}$, it similarly samples $x^*_i \leftarrow\!\!\$\ \{0,1\}^\lambda$ and $\mathsf{ct}^*_{i,j^*-1} \leftarrow \mathsf{Enc}(k, (1, j^*-1, s^*, 0, 0))$.

Then, it samples a constrained public key for the DCS scheme instead of running KeyGen. Specifically, it runs ConstPK with inputs z, ℓ and C_1 as defined in the Share algorithm in Fig. 18, and circuit C_2 such that $C_2(j, x'_j, \mathsf{ct}'_j)$ is 1 if and only if (i) $j \in \mathcal{J} \cup \mathcal{I}$ and (ii) if $j \in \mathcal{I}$, then $\mathsf{ct}'_j = \mathsf{ct}^*_{j,j^*-1}$, and (iii) if $j \in \mathcal{J}$, then, $\mathsf{ct}'_j = \mathsf{ct}_j$, moreover, if $j = j^*$, $x'_j[u] = 1$. Note that it can still generate signatures using sk_{C_2} for shares of all parties in $\mathcal{J} \cup \mathcal{I}$ since they satisfy C_2.

Then, it generates the obfuscated program honestly, and sends it along with $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$ to the adversary. Similar to hybrid H_1 in proof of Lemma 5.7, if the adversary behaves differently in the two hybrids, we can use it to break security of the DCS scheme. This holds crucially because all the signatures seen by the adversary are on messages that satisfy C_2.

Next, we define hybrid H_2, where we change the program P. Specifically, we hard-code the sets $\mathcal{J}$ and $\mathcal{I}$, along with their ciphertexts in P, and the value j^*, position u. Moreover, before checking any signatures, we first set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$, and for input shares of parties in $(\mathcal{J} \cup \mathcal{I})$, we remove them from $\hat{J}$ if the ciphertext in the input share does not match the hard-coded one. Lastly, we also remove j^* from $\hat{J}$ if the input identifier for j^* is not 1 at position u. Figure 21 describes the full program. We argue that this does not change functionality of the program. This is because, for all the parties removed from $\hat{J}$, there is anyway no valid signature on their input shares with respect to the constrained public key.

Let $\mathsf{ct}_i = (r_i, \hat{\mathsf{ct}}_i)$ for all $i \in \mathcal{J}$, and $\mathsf{ct}^*_{i,j^*-1} = (r^*_i, \hat{\mathsf{ct}}^*_{i,j^*-1})$ for all $i \in \mathcal{I}$. We define another hybrid H_3, wherein, we puncture the PRF key k at the values $T = \{r^*_i\}_{i\in\mathcal{I}}$. We hard-code T and the PRF evaluations at these points i.e., $z_r = \mathsf{PRF}'.\mathsf{F}(k, r)$ in the program, and use these to decrypt the ciphertexts for parties in $\mathcal{I}$. Since the functionality of the program is identical in hybrids H_2, H_3, they must be indistinguishable to the adversary, by security of obfuscation.

In hybrid H_4, we now sample the z_r values uniformly randomly – if the r values are unique across ciphertexts of all parties in $\mathcal{J} \cup \mathcal{I}$, then, by security of one-time pad, all the ciphertexts ct^*_{i,j^*-1} are distributed uniformly randomly.

In hybrid H_5, we now change these ciphertexts to encryptions of $(1, j^*-1, s^*, u, 0)$ by simply adding $(0,0,0,u,0)$ to the corresponding $z_{r^*_i}$ value. We argue that this does not change the functionality. To see that, we consider different cases. Note that the only part of the program where the u_i values make a difference is line 8 in Fig. 21, hence we will focus on that in our analysis below.

$\mathsf{P}(\{i, x'_i, \mathsf{ct}'_i, \sigma'_i\}_{i\in J})$:

1. **Constants**: C, k, pk, s, and $\mathcal{J}, \mathcal{I}, \{\mathsf{ct}_i\}_{i\in\mathcal{J}}, \{\mathsf{ct}^*_{i,j^*-1}\}_{i\in\mathcal{J}}, j^*, u$.
2. Set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$.
3. For all $i \in \mathcal{J} \cap J$, if $\mathsf{ct}'_i \neq \mathsf{ct}_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
4. For all $i \in \mathcal{I} \cap J$, if $\mathsf{ct}'_i \neq \mathsf{ct}^*_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
5. If $j^* \in \hat{J}$ and $x'_{j^*}[u] = 0$, then set $\hat{J} \leftarrow \hat{J} \setminus \{j^*\}$.
6. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, x_i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
7. For all $i \in \hat{J}$, let $(f_i, j_i, m_i, u_i, b_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
8. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1 + f_i, \ldots, n\}$. If $f_i = 1$: if $u_i = 0$ or $((j_i + 1) \in J$ and $x_{j_i+1}[u_i] = 1 - b_i)$, then set $S_i \leftarrow S_i \cup \{j_i + 1\}$.
9. Compute $S \leftarrow \bigcap_{i\in\hat{J}} S_i$. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
10. Set $\hat{J} = \hat{J} \cap S$.
11. Abort if $C(x_{\hat{J}}) = 0$.
12. If $\forall\, i \in \hat{J}$, $f_i = 0$, then output s.
13. Otherwise, output the message m_{i^*} where i^* is the smallest index such that $f_i = 1$, i.e., $i^* = \min_{i\in\hat{J}, f_i\neq 0}(i)$.

Figure 21: The modified program P used in hybrid H_2 for proving Lemma 5.23.

- Input $J \subseteq \mathcal{J}$: Since $\mathcal{J} \cap \mathcal{I} = \emptyset$, we have, $J \cap \mathcal{I} = \emptyset$. In this case, changing ciphertexts of parties in $\mathcal{I}$ trivially does not change the program output.
- Input $J \cap \mathcal{I} \neq \emptyset$: Observe that we are changing the ciphertexts for all parties $i \in J \cap \mathcal{I}$ from encrypting $(1, j^* - 1, s^*, 0, 0)$ to $(1, j^* - 1, s^*, u, 0)$. We take two sub-cases:
  - $j^* \notin J$: For each $i \in J \cap \mathcal{I}$, while j^* may or may not be included in the S_i set, since it is not in J, $\hat{J}$ will never have j^* in any case.
  - $j^* \in J$: Observe that if $x'_{j^*}[u]$ were 0 then j^* would be removed by the program even before checking the signatures. Hence we only need to consider the scenario where $x'_{j^*}[u] = 1$. For each $i \in J \cap \mathcal{I}$, the resulting S_i will include j^* in both cases. This is trivially true if ct^*_{i,j^*-1} decrypts to $(1, j^* - 1, s^*, 0, 0)$. In the other case, since $u_i = u$ and $x_{j^*}[u] = 1$, j^* will again be included in S_i.

Observe that hybrid H_5 now corresponds to the box being run with inputs $\{\mathsf{sh}^*_{i,j^*-1,u}\}_{i\in\mathcal{I}}$. Hence, the behavior of the box between hybrids H_0 and H_5 cannot differ by more than $\mathsf{negl}(\lambda)$. This proves the lemma. □

Lemma 5.24. *In the event* GoodBox, *if* $j^* \in \mathcal{J}$, *for all* $u \in [\lambda]$,

$$|\hat{c}_{j^*,0,0} - \hat{c}_{j^*-1,u,0}| \leq \mathsf{negl}(\lambda),$$

if $x_{j^*}[u] = 0$.

Proof. We follow the same strategy as the proof of Lemma 5.23. Specifically, let H_0 denote the hybrid where the challenger runs Share algorithm honestly and runs the box R with inputs $\{\mathsf{sh}^*_{i,j^*}\}_{i\in\mathcal{I}}$.

Hybrid $\hat{H}_0$: We generate signatures included in sh^*_{i,j^*} using the unconstrained signing key sk. By indentical-distribution property of DCS, this hybrid is indistinuighable from H_0.

Hybrid H_1: We sample a constrained public key for the DCS key. Similar to the previous proof, the challenger first samples the identifiers and ciphertexts for parties in $\mathcal{J} \cup \mathcal{I}$, conditioned on $x_{j^*}[u] = 0$. It then runs ConstPK with inputs z, ℓ, C_1, and C_2 defined as in hybrid H_1 above, but for input $(j^*, x'_{j^*}, \mathsf{ct}'_{j^*})$, C_2 is one only if $x'_{j^*}[u] = 0$.

Hybrid H_2: We change the program P, by hard-coding the shares of parties in $\mathcal{J} \cup \mathcal{I}$, along with index j^* and position u. This time, j^* is removed from $\hat{J}$ if the input identifier for j^* is not 0 at position u.

Hybrid H_3: We now puncture the PRF key at the set T as defined in hybrid H_3 above.

Hybrid H_4: We sample the z_r values uniformly randomly. By one-time pad security, the ciphertexts in shares of all parties in $\mathcal{I}$ are uniformly random.

Hybrid H_5: We can now change the ciphertexts in $\mathcal{I}$ to encrypt $(1, j^* - 1, s^*, u, 0)$, by simply adding $(0, -1, 0, u, 0)$ to the $z_{r^*_i}$ values for all $i \in \mathcal{I}$. We argue that the functionality does not change. We take diferent cases:

- $J \subseteq \mathcal{J}$: Changing ciphertexts of parties in $\mathcal{I}$ does not affect the program output.
- $J \cap \mathcal{I} \neq \emptyset$ and $j^* \notin J$: Regardless of whether j^* is included or not in S_i, it will not affect the final $\hat{J}$ set since j^* was not included in the input.
- $J \cap \mathcal{I} \neq \emptyset$ and $j^* \in J$: Observe that if $x'_{j^*}[u]$ where 1 then j^* would already be removed by the program. Hence, we only need to consider the case where $x'_{j^*}[u] = 0$. For any $i \in J \cap \mathcal{I}$, if the ciphertext encrypts $(1, j^*, s^*, 0, 0)$ then the set $S_i = \{j^*+1, \ldots, n\}$. If it encrypts $(1, j^*-1, s^*, u, 0)$, then, since $x'_{j^*}[u] = 0$, j^* will not be included in S_i, resulting in the same set i.e., $S_i = \{j^* + 1, \ldots, n\}$.

Observe that hybrid H_5 now corresponds to the box being run with inputs $\{\mathsf{sh}^*_{i,j^*-1,u}\}_{i\in\mathcal{I}}$. Hence, the behavior of the box between hybrids H_0 and H_5 cannot differ by more than $\mathsf{negl}(\lambda)$. □

Lemmas 5.23 and 5.24 prove that if the jump j^* found in the first part is indeed corrupt, then, the second part of the Trace algorithm will recover the correct pre-image x_{j^*}, except with negligible probability. Specifically, we know that, with overwhelming probability, $\hat{p}_{j^*-1} - \hat{p}_{j^*} > \epsilon/2n$. Let us take two cases:

- $x_{j^*}[u] = 1$. In this case, by Lemma 5.23, we know that $|\hat{p}_{j^*-1} - \hat{c}_{j^*-1,u,0}| \leq \mathsf{negl}(\lambda)$. Since p'_{j^*-1} and $c'_{j^*-1,u}$ are just estimates for the corresponding probabilities $\hat{p}_{j^*-1}$ and $\hat{c}_{j^*-1,u,0}$, by Chernoff bound, we know that $|\hat{c}_{j^*-1,u,0} - c'_{j^*-1,u}| \leq \epsilon/16n$ and $|\hat{p}_{j^*-1} - p'_{j^*-1}| \leq \epsilon/16n$ except with negligible probability. Combining the above, we get that,

$$\begin{aligned} |c'_{j^*-1,u} - p'_{j^*-1}| &= |c'_{j^*-1,u} - \hat{c}_{j^*-1,u,0} + \hat{c}_{j^*-1,u,0} - \hat{p}_{j^*-1} + \hat{p}_{j^*-1} - p'_{j^*-1}| \\ &\leq \epsilon/16n + \mathsf{negl}(\lambda) + \epsilon/16n \\ &\leq 3\epsilon/16n \end{aligned}$$

  Hence, the tracing algorithm will set $x^*_{j^*}[u]$ to be 1, which matches the real identifier except with negligible probability.

- $x_{j^*}[u] = 0$. In this case, by Lemma 5.24, we know that $|\hat{p}_{j^*} - \hat{c}_{j^*-1,u,0}| \leq \mathsf{negl}(\lambda)$. Moreover, by definition of j^*, we know that $\hat{p}_{j^*-1} - \hat{p}_{j^*} \geq \epsilon/2n$. Combining these with Chernoff bounds for p'_{j^*}, p'_{j^*-1} and $c'_{j^*-1,u}$, we get,

$$\begin{aligned} p'_{j^*-1} - c'_{j^*-1,u} &\geq (\hat{p}_{j^*-1} - \epsilon/16n) - (\hat{c}_{j^*-1,u,0} + \epsilon/16n) \\ &\geq (\hat{p}_{j^*-1} - \epsilon/16n) - (\hat{p}_{j^*} + \mathsf{negl}(\lambda) + \epsilon/16n) \\ &\geq \epsilon/2n - \epsilon/8n - \mathsf{negl}(\lambda) \\ &\geq \epsilon/4n \end{aligned}$$

  Hence, in this case, except with negligible probability, we will get $|p'_{j^*-1} - c'_{j^*-1,u}| \geq \epsilon/4n$ and the tracer will set $x^*_{j^*}[u] = 0$ which indeed matches the actual identifier for j^*.

Lastly, we move to the second part, where we prove that the jump j^* found in step 2b of tracing is indeed corrupt. This can be proven essentially by Lemmas 5.9 and 5.10 above. The statement and the proofs of the lemmas are the same, except that all the ciphertexts now encrypt $(\cdot, 0, 0)$. Since the final set returned $\mathcal{J}^*$ is a subset of parties found in the first step, the tracer will never blame an honest party. □

Tracer Non-imputability. Theorem 5.25 below proves tracer non-imputability of our scheme based on the hardness of the one-way function.

Theorem 5.25. *For every* PPT *adversary $\mathcal{A}$, there exists another adversary $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathrm{ni}}_{\mathcal{A},\mathsf{iOGTSSNI}}(\lambda) = \Pr[F(\mathcal{B}(F(x^*))) = F(x^*)],$$

where $x^* \leftarrow\!\!\$\ \{0,1\}^\lambda$ and the probability is over the random coins of $\mathcal{B}$.

Proof of Thm. 5.25. We construct $\mathcal{B}$ to break the one-way function. It gets as input a OWF image y^* from its challenger. It acts as the challenger to $\mathcal{A}$ in the tracer non-imputability game and gets as input (C, i^*, s). It runs the Share algorithm honestly, but only generates shares (and x_i values) of parties $[n] \setminus \{i^*\}$. In the tracing and verification keys, it sets y_{i^*} to y^*, i.e., x_{i^*} is set to be the pre-image of y^*. Note that, even though x_{i^*} is unknown to $\mathcal{B}$, it can stil simulate the adversary's view since x_{i^*} is only used in the share of party i^*, which is not given to $\mathcal{A}$ in this game. It then sends the sampled shares along with tracing key and verification key to $\mathcal{A}$. $\mathcal{A}$ then responds with a set $\mathcal{I}$ containing i^* and a corresponding valid proof π. Then, π must include a tuple $(i^*, x^*_{i^*})$ such that $F(x^*_{i^*}) = y^*$. $\mathcal{B}$ simply sends $x^*_{i^*}$ to its challenger.

It is clear that $\mathcal{B}$ perfectly simulates the tracer non-imputability game to $\mathcal{A}$. This means that $\mathcal{B}$ receives the correct pre-image of y^* with probability equal to advantage of $\mathcal{A}$ in the tracer non-imputability game. This proves the theorem. □

6 A Generic Compiler for GTSS with Strong Traceability

We now present our generic compiler to construct traceable secret sharing schemes with strong traceability, from any secret sharing scheme for monotone circuit access structures SS (without traceability) and a CPA-secure secret-key encryption scheme SKE, which can be instantiated using one-way functions. We refer the reader to [40, Ch. 3] for a formal definition of secret-key encryption and LR-CPA security.

Figure 22 presents the full compiler.

Correctness. Correctness of the compiler follows from correctness of the underlying secret sharing scheme SS. Theorem 6.1 below formally proves this.

$\underline{\mathsf{Share}(1^\lambda, s, C):}$ // $C : \{0,1\}^n \to \{0,1\}$

1. For $i \in [n]$, sample $k_i \leftarrow\!\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$. Run $(s_1, \ldots, s_n, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{SS.Share}(1^\lambda, s, C)$.
2. For all $i \in [n]$, set $\mathsf{sh}_i \leftarrow (k_i, (\hat{c}_i, c_i) \leftarrow\!\!\$\ \mathsf{ShareEnc}(\{k_i\}_{i\in[n]}, s_i, [n], C)$).
3. Output $\{\mathsf{sh}_i\}_{i\in[n]}$, $\mathsf{pp} \leftarrow \mathsf{pp}$, $\mathsf{tk} \leftarrow \{\mathsf{sh}_i\}_{i\in[n]}$, $\mathsf{vk} \leftarrow \bot$.

$\underline{\mathsf{ShareEnc}(\{k_i\}_{i\in[n]}, m, S \subseteq [n], C):}$

1. Sample $\hat{k} \leftarrow\!\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$ and compute $\hat{c} \leftarrow\!\!\$\ \mathsf{SKE.Enc}(\hat{k}, m)$.
2. Sample $(u_1, \ldots, u_n) \leftarrow\!\!\$\ \mathsf{SS.Share}(1^\lambda, \hat{k}, C)$.
3. For all $j \in S$, compute $c_j := \mathsf{SKE.Enc}(k_j, u_j)$.
4. For all $j \in [n] \setminus S$, sample $\hat{u}_j \leftarrow\!\!\$\ \mathcal{SH}_{\mathsf{SS},j,\lambda}$ and compute $c_j := \mathsf{SKE.Enc}(k_j, \hat{u}_j)$. Let $c := \{c_1, \ldots, c_n\}$. Output $(\hat{c}, c)$.

$\underline{\mathsf{ShareDec}(\{k_i\}_{i\in J}, \hat{c}, c):}$

1. Parse c as $(c_1, \ldots, c_n)$. For all $j \in J$, compute $u'_j \leftarrow \mathsf{SKE.Dec}(k_j, c_j)$.
2. Run $\hat{k}' \leftarrow \mathsf{SS.Rec}(\{u_j\}_{j\in J})$. If $\hat{k}' = \bot$, output $\bot$ else output $\mathsf{SKE.Dec}(\hat{k}', \hat{c})$.

$\underline{\mathsf{Rec}(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J}):}$

1. For all $i \in J$, parse sh_i as $(k_i, \hat{c}_i, c_i)$. Compute $s'_i \leftarrow \mathsf{ShareDec}(\{k_i\}_{i\in J}, \hat{c}_i, c_i)$.
2. Output $\mathsf{SS.Rec}(\{s'_i\}_{i\in J})$.

$\underline{\mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I}):}$

1. Parse tk as $\{\mathsf{sh}_i\}_{i\in[n]}$, and for all $i \in [n]$, parse sh_i as $(k_i, \hat{c}_i, c_i)$.
2. Let $J \subseteq [n]$ be any subset such that $C(x_J) = 1$. Let $s \leftarrow \mathsf{Rec}(\{\mathsf{sh}_i\}_{i\in J})$.
3. For $i \in [n]$, compute $\hat{s}_i \leftarrow \mathsf{ShareDec}(\{k_i\}_{i\in J}, \hat{c}_i, c_i)$. Let $\mathcal{J}^* := \bot$.
4. Let $\mathcal{I} = \{i_1, \ldots, i_f\}$. For $j \in \{0, \ldots, f\}$:
   (a) Let $q_j := 0$. For $q \in \{1, \ldots, N\}$: // $N = \frac{8\lambda n^2}{\epsilon^2}$
      i. For $i \in \{i_1, \ldots, i_j\}$: compute $(\hat{c}''_i, c''_i) \leftarrow\!\!\$\ \mathsf{ShareEnc}(\{k_i\}_{i\in[n]}, \hat{s}_i, [n], C)$. Let $\mathsf{sh}''_{i,j} := (k_i, \hat{c}''_i, c''_i)$.
      ii. For $i \in \{i_{j+1}, \ldots, i_f\}$, set $\mathsf{sh}''_{i,j} := \mathsf{sh}_i$.
      iii. Run $b \leftarrow\!\!\$\ R(\{\mathsf{sh}''_{i,j}\}_{i\in\mathcal{I}})$. If $s_b = s$, set $q_j \leftarrow q_j + 1$.
   (b) Compute $q'_j := q_j/N$. If $j > 0$ and $q'_{j-1} - q'_j > \epsilon/4n$, set $\mathcal{J}^* \leftarrow \mathcal{J}^* \cup \{i_j\}$.
5. If $\mathcal{J}^* \neq \bot$, output $\mathcal{J}^*$.
6. Compute $\hat{\epsilon} := \epsilon/2$. For $j \in \{0, \ldots, n\}$:
   (a) Let $p_j := 0$. For $q \in \{1, \ldots, \hat{N}\}$: // $\hat{N} = \frac{8\lambda n^2}{\hat{\epsilon}^2}$
      i. For $i \in \mathcal{I}$, compute $(\hat{c}'_i, c'_i) \leftarrow\!\!\$\ \mathsf{ShareEnc}(\{k_i\}_{i\in[n]}, \hat{s}_i, \mathcal{I} \cup \{j+1, \ldots, n\}, C)$.
      ii. Run $b \leftarrow\!\!\$\ R(\{\mathsf{sh}^*_{i,j} := (k_i, \hat{c}'_i, c'_i)\}_{i\in\mathcal{I}})$. If $s_b = s$, set $p_j \leftarrow p_j + 1$.
   (b) Compute $p'_j := p_j/\hat{N}$. If $j > 0$ and $p_{j-1} - p_j > \hat{\epsilon}/4n$, then set $\mathcal{J}^* \leftarrow \mathcal{J}^* \cup \{j\}$.
7. Output $\mathcal{J}^*$.

Figure 22: Our generic compiler Π_{GTSS} to construct TSS from any secret sharing scheme SS for monotone circuits and any secret-key encryption scheme SKE. Here, $\mathcal{SH}_{\mathsf{SS},j,\lambda}$ denotes the space of secret shares of party $j \in [n]$ for SS.

Theorem 6.1. *The scheme Π_{GTSS} is correct with negligible error, if the secret sharing scheme SS has negligible correctness error.*

Proof of Thm. 6.1. Let us consider any access structure C and set $J \subseteq [n]$ such that $C(x_J) = 1$. Let us parse sh_i as $(k_i, \hat{c}_i, c_i)$, and c_i as $(c_{i,1}, \ldots, c_{i,n})$ for all $i \in J$. For any $i, j \in J$, the ciphertext $c_{i,j}$ is an

encryption of $u_{i,j}$ with party j's key k_j, which is one share of the key $\hat{k}_i$ (the key sampled during $\mathsf{ShareEnc}$ for encrypting s_i). This means that the decryption $u'_{i,j} = u_{i,j}$ for all $j \in J$ as the honest shares contain encryptions to the set $[n]$. Next, since $C(x_J) = 1$, running $\mathsf{SS.Rec}$ on the shares $\{u_{i,j}\}$ will indeed output the reconstructed secret $\hat{k}_i$ by correctness of SS. Formally, we get $\hat{k}'_i = \hat{k}_i$ for all $i \in J$.

Similarly, for all $i \in J$, $\hat{c}_i$ is an encryption of s_i with respect to $\hat{k}_i$. Hence, we have $s'_i = s_i$. Recall that s_i is the ith secret share of s with respect to SS. Hence, running $\mathsf{SS.Rec}$ on $\{s_i\}_{i\in J}$ will indeed output the shared secret s, by correctness of SS. □

Secrecy. Secrecy of the compiler directly follows from secrecy of the underlying secret sharing scheme SS. We prove this in Theorem 6.2.

Theorem 6.2. *For all* PPT *adversaries* $\mathcal{A}$*, there exists another* PPT *adversary* $\mathcal{B}$ *such that,*

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathcal{A},\Pi_{\mathsf{GTSS}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B},\mathsf{SS}}(\lambda)$$

Proof of Thm. 6.2. We will construct $\mathcal{B}$ which plays the secrecy game for SS, and acts as the challenger to $\mathcal{A}$ in the secrecy game for Π_{GTSS}. For any secrets s_0, s_1, any circuit C and any subset $\mathcal{J} \subseteq [n]$ such that $C(x_{\mathcal{J}}) = 0$, $\mathcal{B}$ gets from its challenger the shares for parties in $\mathcal{J}$ i.e., $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$ along with public information pp. It then samples keys $k_1, \ldots, k_n \leftarrow\!\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$, and honestly computes the shares of parties in $\mathcal{J}$ for Π_{GTSS} by running $(\hat{c}_i, c_i) \leftarrow\!\!\$\ \mathsf{ShareEnc}(\{k_i\}_{i\in[n]}, \mathsf{sh}_i, [n])$ for all $i \in \mathcal{J}$. It then sends pp along with $\mathsf{sh}'_i \leftarrow (k_i, \hat{c}_i, c_i)$ to $\mathcal{A}$, and simply forwards $\mathcal{A}$'s output to its challenger.

It can be seen that $\mathcal{B}$ perfectly simulates the secrecy game to $\mathcal{A}$. Moreover, if $\mathcal{A}$ guesses the bit b correctly, then $\mathcal{B}$ also wins its secrecy game. □

Strong Traceability. We prove strong traceability of the compiler based on CPA security of the encryption scheme and secrecy of the secret sharing scheme SS.

Theorem 6.3. *For all* PPT *adversaries* $\mathcal{A}$*, if* SS *has secrecy and* SKE *achieves LR-CPA security, then,*

$$\mathsf{Adv}^{\mathrm{str-trac}}_{\mathcal{A},\Pi_{\mathsf{GTSS}}}(\lambda) \leq \mathsf{negl}(\lambda)$$

Proof of Thm. 6.3. We start with some notation. Let us use s_0, s_1 to denote the secrets given by $\mathcal{A}$. Let $s = s_b$ denote the secret that is shared in the Share algorithm, where b is sampled uniformly at random from $\{0, 1\}$. Let $\mathcal{I}$ denote the label of the reconstruction box R, and let $\mathcal{J}$ denote the set of corrupt parties. Let E_1 denote the event that the tracer finds no corrupt party in step 4.

We prove the theorem in three parts. Lemma 6.4 proves that the tracer will never blame an honest party in step 4. Second, in the event E_1 where the tracer finds no parties in this step, we prove in Lemma 6.5 that the tracer must find at least one party in step 6 if the box is minimally-useful. Lastly, Lemma 6.6 proves that this party must be corrupt.

For all $j \in \{0, \ldots, f\}$, let $\hat{q}_j$ denote the probability that the box R outputs the correct bit b when run with shares $\{\mathsf{sh}''_{i,j}\}_{i\in\mathcal{I}}$ as defined in Fig. 22. Observe that the q'_j values computed in the Trace algorithm are essentially approximations of the probability $\hat{q}_j$. We are now ready for the first lemma.

Lemma 6.4. *For all* $j \in [f]$ *such that* $i_j \in \mathcal{I} \setminus \mathcal{J}$*,*

$$q'_{j-1} - q'_j \leq \epsilon/8n,$$

except with negligible probability.

Proof. For any honest party $i_j \in \mathcal{I} \setminus \mathcal{J}$, observe that the view of the adversary contains no information about the real share sh_{i_j} of this party. Moreover, the share $\mathsf{sh}''_{i_j,j}$ sampled in the Trace algorithm is distributed identically as the real share sh_{i_j} – both contain the same key k_{i_j}, and an encryption of the same value s_i obtained by running ShareEnc algorithm with the same inputs i.e., $\{k_i\}_{i\in[n]}, s_i$ and $[n]$. Lastly, observe that for any $i \in \{i_1, \ldots, i_{j-1}\}$, the shares $\mathsf{sh}''_{i,j-1}$ and $\mathsf{sh}''_{i,j}$ are sampled from the same distribution. More formally, for an adversary that only corrupts $\mathcal{J}$, we have,

$$\{\mathsf{sh}''_{i_1,j-1}, \ldots, \mathsf{sh}''_{i_{j-1},j-1}, \mathsf{sh}_{i_j}, \ldots, \mathsf{sh}_{i_f}\} \approx_s \{\mathsf{sh}''_{i_1,j}, \ldots, \mathsf{sh}''_{i_j,j}, \mathsf{sh}_{i,j+1} \ldots, \mathsf{sh}_{i_f}\}.$$

This means that $\hat{q}_{j-1} = \hat{q}_j$. In other words, the box must behave identically when run with shares $\{\mathsf{sh}''_{i,j-1}\}$ and $\{\mathsf{sh}''_{i,j}\}$.

Next, by Chernoff bound, we know that for any $j \in \{0, \ldots, f\}$, $|\hat{q}_j - q'_j| \leq \epsilon/16n$, with probability $\geq 1 - 2e^{-\lambda/16}$. Hence, we get:

$$\begin{aligned} q'_{j-1} - q'_j &= q'_{j-1} - \hat{q}_{j-1} + \hat{q}_{j-1} - \hat{q}_j + \hat{q}_j - q'_j && (1)\\ &\leq \epsilon/16n + 0 + \epsilon/16n. && (2)\end{aligned}$$

□

Lemma 6.5. *1. If the events* GoodBox *and* E_1 *occur, then, with overwhelming probability,*

$$\Pr[R(\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}) = b] \geq \frac{1}{2} + \hat{\epsilon}$$

2. In the event GoodBox*, then, there exist* PPT *adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}$ *such that*

$$\Pr[R(\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}) = b] \leq \frac{1}{2} + 2(n-1) \cdot \left(\mathsf{Adv}^{\mathsf{sec}}_{\mathcal{B},\mathsf{SS}}(\lambda) + \mathsf{Adv}^{\mathsf{ss}}_{\mathcal{C},\mathsf{SKE}}(\lambda)\right) + \mathsf{Adv}^{\mathsf{sec}}_{\mathcal{D},\mathsf{SS}}(\lambda).$$

Proof. For the first part, recall that the event E_1 occurs if no party was found in step 4. In the event GoodBox, R must be a minimally-useful box and $C(x_{\mathcal{J}}) = C(x_{\mathcal{I}}) = 0$. This means that $\hat{q}_0 = 1/2 + \epsilon$, as the shares $\{\mathsf{sh}''_{i,0}\}_{i\in\mathcal{I}}$ are the real shares of $\mathcal{I}$. Moreover, in the event E_1, we have $q'_{j-1} - q'_j \leq \epsilon/4n$ for all $j \in [f]$. Adding this equation for all $j \in [f]$, we get $q'_0 - q'_f \leq (\epsilon/4n) \cdot |\mathcal{I}| \leq \epsilon/4$. Additionally, by Chernoff bound, $|\hat{q}_j - q'_j| \leq \epsilon/16n$ for $j \in \{0, f\}$ except with probability $(1 - 2e^{-\lambda/16})$. Combining all these equations together, we get,

$$\begin{aligned} \hat{q}_f &\geq q'_f - \epsilon/16n && (3)\\ &\geq q'_0 - \epsilon/4 - \epsilon/16n && (4)\\ &\geq \hat{q}_0 - \epsilon/16n - \epsilon/4 - \epsilon/16n && (5)\\ &\geq 1/2 + \epsilon - \epsilon/16n - \epsilon/4 - \epsilon/16n && (6)\\ &\geq 1/2 + \epsilon/2 && (7)\end{aligned}$$

Informally, this implies that the box gives the correct output b with probability $1/2 + \hat{\epsilon}$ when given as input freshly sampled shares for parties in $\mathcal{I}$. We finish the proof by observing that the shares $\{\mathsf{sh}''_{i,f}\}_{i\in\mathcal{I}}$ and $\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}$ are distributed identically – for each $i \in \mathcal{I}$, both $\mathsf{sh}''_{i,f}$ and $\mathsf{sh}^*_{i,0}$ contain the real key k_i, and a freshly sampled encryption of share $\hat{s}_i$ obtained by running ShareEnc with inputs $\{k_i\}_{i\in[n]}, \hat{s}_i$ and the set $[n]$. This proves the first part.

For the second part, we will consider a series of hybrids. Let H_0 denote the original game where the tracer runs the box R with inputs $\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}$. For all $i \in \mathcal{I}$, let $\mathsf{sh}^*_{i,n} = (k_i, \hat{c}_i, c_i)$. Note that, for all $i \in \mathcal{I}$, these $\hat{c}_i, c_i$ are generated by running ShareEnc algorithm with inputs $\{k_i\}_{i\in[n]}$, the secret share

$\hat{s}_i$ and the set $\mathcal{I}$. Let $\hat{k}_i$ denote the symmetric key sampled to produce these ciphertexts. Formally, $\hat{c}_i = \mathsf{SKE.Enc}(\hat{k}_i, \hat{s}_i)$. Moreover, let $c_i = c_{i,1}, \ldots, c_{i,n}$, where $c_{i,j}$ is an encryption of the jth share of $\hat{k}_i$ denoted by $u_{i,j}$ for all $j \in \mathcal{I}$, and an encryption of a randomly sampled value $\hat{u}_{i,j}$ for all $j \in [n] \setminus \mathcal{I}$. In other words, the ciphertext c_i only contains encryptions of shares of $\hat{k}_i$ for parties in $\mathcal{I}$. Lastly, let $\mathcal{I} = \{i_1, \ldots, i_f\}$.

We now define another hybrid H_1 where, for i_1, the challenger samples the ciphertext $(\hat{c}_{i_1}, c_{i_1})$ slightly differently. Specifically, it samples $\hat{k}_{i_1}$ as before, and computes $\hat{c}_{i_1}$ by encrypting $\hat{s}_{i_1}$ with $\hat{k}_{i_1}$, but, instead of sampling shares of $\hat{k}_{i_1}$, it samples another key $\hat{k}'_{i_1} \leftarrow\!\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$ and computes its shares $(u'_{i_1,1}, \ldots, u'_{i_1,n})$. Then, for all j in $[n] \setminus \mathcal{I}$, it still encrypts random values $\hat{u}_{i_1,j} \leftarrow\!\!\$\ \mathcal{SH}_{\mathsf{SS},j,\lambda}$ as in hybrid H_0, but for $j \in \mathcal{I}$, it encrypts $u'_{i_1,j}$ (instead of $u_{i_1,j}$ as in hybrid H_0). We argue that, in the event $\mathsf{GoodBox}$ i.e., $C(x_{\mathcal{I}}) = 0$, if the box behaves differently in hybrids H_0 and H_1, we can use $\mathcal{A}$ to break secrecy of the secret-sharing scheme SS.

Formally, we will construct an adversary $\mathcal{B}_1$ playing the secrecy game for SS, that will act as the challenger to $\mathcal{A}$. $\mathcal{B}_1$ gets from $\mathcal{A}$ the access structure, the set of corrupt parties and two values: $C, \mathcal{J}, s_0, s_1$. It samples $b \leftarrow\!\!\$\ \{0,1\}$ and runs Share algorithm as the honest challenger in the strong traceability game, and sends $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$ to $\mathcal{A}$. It then gets the box R and the label $\mathcal{I}$ from $\mathcal{A}$. It samples shares $\{\mathsf{sh}^*_{i,n}\}_{i \in \mathcal{I}}$ as follows: for $i \in \{i_2, \ldots, i_f\}$, it samples the shares exactly as the tracer, i.e., $\hat{c}_i, c_i \leftarrow\!\!\$\ \mathsf{ShareEnc}(\{k_i\}_{i \in [n]}, \hat{s}_i, \mathcal{I})$. But for i_1, it samples $\hat{k}_{i_1}$ as in the $\mathsf{ShareEnc}$ algorithm and computes $\hat{c}_{i_1}$ by encrypting $\hat{s}_{i_1}$. It then samples another key $\hat{k}'_{i_1}$, and sends $\hat{k}_{i_1}$ and $\hat{k}'_{i_1}$, along with the set $\mathcal{I}$ and C to its secrecy challenger. This is reasonable since $\mathcal{I}$ is an unauthorized set in the event $\mathsf{GoodBox}$. $\mathcal{B}_1$ gets back some secret shares from its challenger, denoted by $u^*_{i_1}, \ldots, u^*_{i_f}$, where $\mathcal{I} = \{i_1, \ldots, i_f\}$. $\mathcal{B}_1$ then computes c_{i_1} as follows: for all $j \in [n] \setminus \mathcal{I}$, $c_{i_1,j} \leftarrow\!\!\$\ \mathsf{SKE.Enc}(k_j, \hat{u}_{i_1,j})$ where $\hat{u}_{i_1,j} \leftarrow\!\!\$\ \mathcal{SH}_{\mathsf{SS},j,\lambda}$ is sampled randomly. For all $j \in \mathcal{I}$, $c_{i_1,j} \leftarrow\!\!\$\ \mathsf{SKE.Enc}(k_j, u^*_j)$. Observe that if $\mathcal{B}_1$ gets shares of $\hat{k}_{i_1}$ from its challenger, then $\mathcal{B}_1$ perfectly simulates hybrid H_0, and if it gets shares of $\hat{k}'_{i_1}$, it simulates hybrid H_1. Lastly, $\mathcal{B}_1$ runs R with the sampled shares for $\mathcal{I}$. If the box outputs b, $\mathcal{B}_1$ returns 0, otherwise 1. Let P_i denote the probability that R outputs the correct bit b in hybrid H_i, for $i \in \{0,1\}$. Then, we get that,

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1,\mathsf{SS}}(\lambda) = \frac{1}{2}|P_0 - P_1|.$$

Generalizing the above, we can define hybrids $H_2, \ldots, H_f$, where in hybrid H_w, the ciphertexts in shares of $i_1, \ldots, i_w$ are sampled as in hybrid H_1 as described above, and the shares $\mathsf{sh}^*_{i_{w+1}}, \ldots, \mathsf{sh}^*_{i_f}$ are sampled as in the original game H_0. By a similar reduction as above, for every $i \in [f]$, there exists a PPT adversary $\mathcal{B}_i$ such that

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_i,\mathsf{SS}}(\lambda) = \frac{1}{2}|P_{i-1} - P_i|,$$

where P_i denote the probability that R outputs the correct bit b in hybrid H_i, for $i \in \{0, \ldots, f\}$. Combining all the $\mathcal{B}_i$ into a single adversary that chooses i at random, we get that,

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B},\mathsf{SS}}(\lambda) \geq \frac{|P_0 - P_f|}{2f}$$

We now make another series of hybrids. We define hybrid $H_{f+1}, \ldots, H_{2f}$, where in hybrid H_{f+w}, the ciphertexts $\hat{c}_{i_1}, \ldots, \hat{c}_{i_w}$ are sampled slightly differently. Specifically, for all $i \in \{i_1, \ldots, i_w\}$, instead of encrypting $\hat{s}_i$, the challenger samples $\hat{t}_i \leftarrow\!\!\$\ \mathcal{SH}_{\mathsf{SS},i,\lambda}$ and computes $\hat{c}_i \leftarrow\!\!\$\ \mathsf{SKE.Enc}(\hat{k}_i, \hat{t}_i)$. We claim that if the box behaves differently in hybrids H_{f+w-1} and H_{f+w}, we can use $\mathcal{A}$ to break semantic security of the encryption scheme SKE. More formally, we can construct an adversary $\mathcal{C}_w$ for semantic security. $\mathcal{C}_w$ acts as the challenger to $\mathcal{A}$, it does everything as the honest challenger until it gets the box R and the label $\mathcal{I}$ from $\mathcal{A}$. It then samples shares of parties in $\mathcal{I}$ to run the box, as follows. For all $i \in \{i_{w+1}, \ldots, i_f\}$, the share $\mathsf{sh}^*_{i,n}$ is sampled exactly as in hybrid H_f. For all $i \in \{i_1, \ldots, i_{w-1}\}$, it samples $\hat{k}_i$, $\hat{k}'_i$ and c_i as

in H_f, and computes $\hat{c}_i \leftarrow\!\!\$\ \mathsf{SKE.Enc}(\hat{k}_i, \hat{t}_i)$ where $\hat{t}_i$ is sampled uniformly at random. Lastly, for i_w, it samples $\hat{t}_{i_w} \leftarrow\!\!\$\ \mathcal{SH}_{\mathsf{SS},i_w,\lambda}$ and $\hat{k}'_{i_w}$, and computes c_{i_w} as in hybrid H_f. Notably, it does not sample $\hat{k}_{i_w}$, and essentially uses its challenger for producing $\hat{c}_{i_w}$ as follows. It sends $\hat{s}_{i_w}$ and $\hat{t}_{i_w}$ to its challenger, and gets back a ciphertext c^*. It sets $\hat{c}_{i_w} \leftarrow c^*$, and runs the box R with these shares. Observe that if $\mathcal{C}_w$'s challenger sent an encryption of $\hat{s}_{i_w}$, then $\mathcal{C}_w$ perfectly simulates the hybrid H_{f+w-1}, and if c^* is an encryption of $\hat{t}_{i_w}$, then $\mathcal{C}_w$ simulates the hybrid H_{f+w}. Lastly, if the box R outputs b when run with these shares, $\mathcal{C}_w$ outputs 0 and otherwise it outputs 1. Let P_i denote the probability that R outputs the correct bit b in hybrid H_i, for $i \in \{0, \ldots, 2f\}$. Then, by a simple analysis, we get that $\mathcal{C}_w$ can break semantic security of P_{w-1} and P_w differ by a non-negligible amount. More formally,

$$\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{C}_w,\mathsf{SKE}}(\lambda) = \frac{|P_{f+w-1} - P_{f+w}|}{2}$$

Combining all the $\mathcal{C}_w$ into a single adversary $\mathcal{C}$ which samples $w \leftarrow\!\!\$\ [f]$, we get,

$$\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{C},\mathsf{SKE}}(\lambda) \geq \frac{|P_f - P_{2f}|}{2f}.$$

Lastly, observe that in hybrid H_{2f}, the challenger only needs the secret shares of s_b for parties in $\mathcal{J}$, and in the event $\mathsf{GoodBox}$, $C(x_{\mathcal{J}}) = 0$. This allows us to prove that if the box R outputs b with (half plus) non-negligible probability, then, we can use $\mathcal{A}$ to break secrecy of SS. More formally, we will construct a PPT adversary $\mathcal{D}$ that acts as the challenger to $\mathcal{A}$. It gets as input s_0, s_1, the set $\mathcal{J}$ along with the access structure C. It forwards these things to its challenger. Here, we crucially rely on the fact that $C(x_{\mathcal{J}}) = 0$. It gets back shares s_j for all $j \in \mathcal{J}$. It then samples keys $k_1, \ldots, k_n \leftarrow\!\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$ and computes sh_j for all $j \in \mathcal{J}$ as the honest challenger, and sends these shares to $\mathcal{A}$. It then gets back a box R and the label $\mathcal{I}$. It samples shares $\mathsf{sh}^*_{i,n}$ for all parties in $\mathcal{I}$ exactly as in hybrid H_{2f} – crucially, it does not need real shares of the secret s_b for these parties. Formally, for all $i \in \mathcal{I}$, it samples $\hat{t}_i$, and keys $\hat{k}_i, \hat{k}'_i$, and computes $\hat{c}_i$ as an encryption of $\hat{t}_i$, and computes c_i with encryptions of random values for parties $j \in [n] \setminus \mathcal{I}$, and encryptions of shares of $\hat{k}'_i$ for parties $j \in \mathcal{I}$. Observe, that if the secrecy challenger chose secret $s_{\hat{b}}$, then $\mathcal{D}$ perfectly simulates the hybrid H_{2f} with $b = \hat{b}$ to $\mathcal{A}$. Lastly, $\mathcal{D}$ simply forwards the output of R to its challenger. Observe that, $\mathcal{D}$ will output the correct bit $\hat{b}$ with the same probability as R outputting b. Hence,

$$\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{D},\mathsf{SKE}}(\lambda) = \left| P_{2f} - \frac{1}{2} \right|.$$

Combining all the above equations and the fact that $f < n$, we get the lemma. □

For all $j \in \{0, \ldots, n\}$, let $\hat{p}$ denote the probability with which the box R outputs b when run with shares $\{\mathsf{sh}^*_{i,j}\}_{i \in \mathcal{I}}$. The above lemma proves that $\hat{p}_0 \geq 1/2 + \hat{\epsilon}$ and $\hat{p}_n \leq 1/2 + \mathsf{negl}(\lambda)$ if the SS scheme has secrecy and SKE scheme has semantic security. Combining the two, we get that there must be a jump in these probabilities for some $j \in [n]$. Formally, there must exist at least one $j^* \in [n]$ such that $\hat{p}_{j^*-1} - \hat{p}_{j^*} \geq \hat{\epsilon}/2n$.

Next, observe that the values p'_j computed in the Trace algorithm are an approximation of the probabilities $\hat{p}_j$. By Chernoff bound, we get that,for all $j \in \{0, \ldots, n\}$,

$$\Pr[|\hat{p}_j - p'_j| > \frac{\hat{\epsilon}}{16n}] < 2e^{-\lambda/64}.$$

Hence, we get that, for j^*, $p'_{j^*-1} - p'_{j^*} \geq \hat{\epsilon}/2n - \hat{\epsilon}/8n > \hat{\epsilon}/4n$. This proves that if the events $\mathsf{GoodBox}$ and E_1 occur, then the Trace algorithm will blame at least one party i.e., j^*. Hence, $\mathcal{J}^* \neq \perp$, except with negligible probability in the event $\mathsf{GoodBox} \wedge \mathsf{E}_1$.

We now move to the last part, where we prove that the jump found by the tracer can not be honest.

Lemma 6.6. *1. For all $j \in \mathcal{I} \setminus \mathcal{J}$, $\hat{p}_{j-1} = \hat{p}_j$.*

2. For all $j \in [n] \setminus (\mathcal{J} \cup \mathcal{I})$, there exists a PPT *adversary $\mathcal{B}$ such that,*

$$|\hat{p}_j - \hat{p}_{j-1}| \leq 2 \cdot \mathsf{Adv}^{\mathrm{lr-cpa}}_{\mathcal{B},\mathsf{SKE}}(\lambda).$$

Proof. For the first part, consider any $j \in \mathcal{I} \setminus \mathcal{J}$, and let us analyze the shares $\mathsf{sh}^*_{i,j-1}$ and $\mathsf{sh}^*_{i,j}$ sampled by the tracer. Observe that the shares $\mathsf{sh}^*_{i,j-1}$ and $\mathsf{sh}^*_{i,j}$ are computed using the $\mathsf{ShareEnc}$ algorithm with the keys $\{k_i\}$, the share $\hat{s}_i$, and the only difference is the set. For $\mathsf{sh}^*_{i,j-1}$, the set input to $\mathsf{ShareEnc}$ is $\mathcal{I} \cup \{j, \ldots, n\}$. For $\mathsf{sh}^*_{i,j}$, the set input to $\mathsf{ShareEnc}$ is $\mathcal{I} \cup \{j+1, \ldots, n\}$. But observe that since $j \in \mathcal{I}$, both these sets are actually equal – this means that the shares $\mathsf{sh}^*_{i,j-1}$ and $\mathsf{sh}^*_{i,j}$ as sampled by the tracer come from the same distribution. Hence, the behavior of the box must also be identical in the two cases. This proves the first part.

For the second part, we will rely on IND-CPA semantic security of the encryption scheme SKE. Formally, we will construct an adversary $\mathcal{B}$ using $\mathcal{A}$, that can break IND-CPA for SKE. $\mathcal{B}$ gets s_0, s_1, the access structure C and the set of corrupt parties $\mathcal{J}$ from $\mathcal{A}$. It samples $b \leftarrow\!\$\ \{0,1\}$, and computes secret shares of s_b, let us use $s_{b,1}, \ldots, s_{b,n}$ to denote the output of $\mathsf{SS.Share}(1^\lambda, s_b, C)$. It then samples the keys $k_i \leftarrow\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$ for all parties $i \in [n] \setminus \{j\}$ except for j.

$\mathcal{B}$ then samples shares of all parties in $\mathcal{J}$ as follows. For each $i \in \mathcal{J}$, it essentially runs the $\mathsf{ShareEnc}$ algorithm, but queries its challenger for any ciphertext encrypted to party j. More formally, for each $i \in \mathcal{J}$, $\mathcal{B}$ samples a key $\hat{k}$, and computes $\hat{c}$ by encrypting $s_{b,i}$ using $\hat{k}$. It then computes shares $u_1, \ldots, u_n$ of $\hat{k}$, and for all $w \in [n] \setminus \{j\}$, computes $c_w \leftarrow\!\$\ \mathsf{SKE.Enc}(k_w, u_w)$. Lastly, to compute c_j, $\mathcal{B}$ queries its CPA challenger for an encryption to u_j. $\mathcal{B}$ then sends $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$ to $\mathcal{A}$.

$\mathcal{B}$ will get a reconstruction box R along with a label $\mathcal{I}$ from $\mathcal{A}$. It then samples shares for $\mathcal{I}$ that it will use to run the box. For each $i \in \mathcal{I}$, note that $\mathcal{B}$ has sampled the key k_i since $j \notin \mathcal{I}$, so the only thing $\mathcal{B}$ needs to do is to compute the ciphertexts in the shares of these parties. For each $i \in \mathcal{I}$, it samples a key $\hat{k} \leftarrow\!\$\ \mathsf{SKE.KeyGen}(1^\lambda)$ and computes $\hat{c}$ by encrypting s_i using $\hat{k}$, as in the $\mathsf{ShareEnc}$ algorithm. It then computes shares $(u_1, \ldots, u_n)$ of $\hat{k}$. For all parties $w \in \mathcal{I} \cup \{j+1, \ldots, n\}$, $\mathcal{B}$ computes $c_w \leftarrow\!\$\ \mathsf{Enc}(k_w, u_w)$. For all parties $w \in [n] \setminus (\mathcal{I} \cup \{j, \ldots, n\})$, $\mathcal{B}$ samples a random share $\hat{u}_w \leftarrow\!\$\ \mathcal{SH}_{\mathsf{SS},w,\lambda}$ and encrypts it to get $c_w \leftarrow\!\$\ \mathsf{SKE.Enc}(k_w, \hat{u}_w)$. Lastly, to compute c_j, it sends to its challenger $(u_j, \hat{u}_j)$ as the challenge messages, where $\hat{u}_j \leftarrow\!\$\ \mathcal{SH}_{\mathsf{SS},j,\lambda}$, and gets back a ciphertext c^*. It sets $c_j = c^*$. Note that $\mathcal{B}$ is allowed to send multiple pairs of challenge messages since we are considering the LR-CPA security game as defined in [40].

$\mathcal{B}$ then runs the box R with the shares of $\mathcal{I}$ sampled above. If R outputs b then $\mathcal{B}$ returns 0 to its challenger otherwise 1. Observe that, if the LR-CPA challenger encrypts the left messages, then $\mathcal{B}$ perfectly simulates the shares $\{\mathsf{sh}^*_{i,j-1}\}_{i \in \mathcal{I}}$, whereas if the challenger encrypts the right messages, then the simulated shares are distributed identically as $\{\mathsf{sh}^*_{i,j}\}_{i \in \mathcal{I}}$. This means that the advantage of $\mathcal{B}$ is $|\hat{p}_{j-1} - \hat{p}_j|/2$. This proves the lemma.

□

Lastly, recall that p'_j is an approximation of $\hat{p}_j$. By Chernoff bound, for all $j \in \{0, \ldots, n\}$, $|p'_j - \hat{p}_j| \leq \hat{\epsilon}/16n$ except with probability $2e^{-\lambda/16}$. Combining this with the above lemma, we see that for any honest party j, $|p'_{j-1} - p'_j| < \hat{\epsilon}/4n$ if SKE achieves LR-CPA security. □

Acknowledgements. Part of this work was done while the authors were visiting the Simons Institute for the Theory of Computing.

References

[1] B. Applebaum, A. Beimel, Y. Ishai, E. Kushilevitz, T. Liu, and V. Vaikuntanathan. Succinct computational secret sharing. In B. Saha and R. A. Servedio, editors, *55th ACM STOC*, pages 1553–1566, Orlando, FL, USA, June 20–23, 2023. ACM Press.

[2] K. Baghery, E. Ebrahimi, O. Mirzamohammadi, and M. Sedaghat. Traceable verifiable secret sharing and applications. Cryptology ePrint Archive, Report 2025/318, 2025.

[3] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18, Santa Barbara, CA, USA, Aug. 19–23, 2001. Springer Berlin Heidelberg, Germany.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[5] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In S. Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 27–35, Santa Barbara, CA, USA, Aug. 21–25, 1990. Springer, New York, USA.

[6] M. Bertilsson and I. Ingemarsson. A construction of practical secret sharing schemes using linear block codes. In J. Seberry and Y. Zheng, editors, *AUSCRYPT'92*, volume 718 of *LNCS*, pages 67–79, Gold Coast, Queensland, Australia, Dec. 13–16, 1993. Springer Berlin Heidelberg, Germany.

[7] A. Bishop, M. Green, Y. Ishai, A. Jain, and P. Lou. Fully anonymous secret sharing. In *Advances in Cryptology – CRYPTO 2025: 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part IV*, page 356–389, Berlin, Heidelberg, 2025. Springer-Verlag.

[8] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.

[9] D. Boneh and M. K. Franklin. An efficient public key traitor tracing scheme. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 338–353, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer Berlin Heidelberg, Germany.

[10] D. Boneh, A. Partap, and L. Rotem. Accountability for misbehavior in threshold decryption via threshold traitor tracing. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 317–351, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.

[11] D. Boneh, A. Partap, and L. Rotem. Traceable secret sharing: Strong security and efficient constructions. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part V*, volume 14924 of *LNCS*, pages 221–256, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.

[12] D. Boneh, A. Sahai, and B. Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 573–592, St. Petersburg, Russia, May 28 – June 1, 2006. Springer Berlin Heidelberg, Germany.

[13] D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499, Santa Barbara, CA, USA, Aug. 17–21, 2014. Springer Berlin Heidelberg, Germany.

[14] J. Bormet, S. Dziembowski, S. Faust, T. Lizurej, and M. Mielniczuk. Strong secret sharing with snitching. Cryptology ePrint Archive, Paper 2025/1119, 2025.

[15] H. Chabanne, D. H. Phan, and D. Pointcheval. Public traceability in traitor tracing schemes. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 542–558, Aarhus, Denmark, May 22–26, 2005. Springer Berlin Heidelberg, Germany.

[16] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[17] Y. Chen, V. Vaikuntanathan, B. Waters, H. Wee, and D. Wichs. Traitor-tracing from LWE made simple and attribute-based. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 341–369, Panaji, India, Nov. 11–14, 2018. Springer, Cham, Switzerland.

[18] Y. Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127, Santa Barbara, CA, USA, Aug. 16–20, 1988. Springer Berlin Heidelberg, Germany.

[19] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315, Santa Barbara, CA, USA, Aug. 20–24, 1990. Springer, New York, USA.

[20] L. Devadas, A. Jain, B. Waters, and D. J. Wu. Succinct witness encryption for batch languages and applications. ASIACRYPT, 2025.

[21] X. T. Do, D. H. Phan, and D. Pointcheval. Traceable inner product functional encryption. In S. Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 564–585, San Francisco, CA, USA, Feb. 24–28, 2020. Springer, Cham, Switzerland.

[22] Y. Dodis and N. Fazio. Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 100–115, Miami, FL, USA, Jan. 6–8, 2003. Springer Berlin Heidelberg, Germany.

[23] S. Dziembowski, S. Faust, T. Lizurej, and M. Mielniczuk. Secret sharing with snitching. In B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, editors, *ACM CCS 2024*, pages 840–853, Salt Lake City, UT, USA, Oct. 14–18, 2024. ACM Press.

[24] O. Farràs and M. Guiot. Traceable secret sharing schemes for general access structures. Cryptology ePrint Archive, Paper 2025/1120, 2025.

[25] A. Fiat and T. Tassa. Dynamic traitor training. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 354–371, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer Berlin Heidelberg, Germany.

[26] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49, Berkeley, CA, USA, Oct. 26–29, 2013. IEEE Computer Society Press.

[27] S. Garg, A. Kumarasubramanian, A. Sahai, and B. Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 2010*, pages 121–130, Chicago, Illinois, USA, Oct. 4–8, 2010. ACM Press.

[28] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[29] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[30] J. Gong, J. Luo, and H. Wee. Traitor tracing with $N^{1/3}$-size ciphertexts and $O(1)$-size keys from k-Lin. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 637–668, Lyon, France, Apr. 23–27, 2023. Springer, Cham, Switzerland.

[31] R. Goyal, V. Koppula, and B. Waters. Collusion resistant traitor tracing from learning with errors. In I. Diakonikolas, D. Kempe, and M. Henzinger, editors, *50th ACM STOC*, pages 660–670, Los Angeles, CA, USA, June 25–29, 2018. ACM Press.

[32] R. Goyal, V. Koppula, and B. Waters. New approaches to traitor tracing with embedded identities. In D. Hofheinz and A. Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 149–179, Nuremberg, Germany, Dec. 1–5, 2019. Springer, Cham, Switzerland.

[33] R. Goyal and B. Waters. Personal communication, 2025.

[34] V. Goyal, Y. Song, and A. Srinivasan. Traceable secret sharing and applications. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 718–747, Virtual Event, Aug. 16–20, 2021. Springer, Cham, Switzerland.

[35] C. Hoffmann. Traceable secret sharing based on the chinese remainder theorem. Cryptology ePrint Archive, Report 2024/811, 2024.

[36] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. IEEE Global Telecommunication Conf. (Globecom'87)*, pages 99–102, 1987.

[37] A. Jain, Z. Jin, S. Mathialagan, and O. Paneth. On succinct obfuscation via propositional proofs (or: How to use pv-io). To appear at STOC 2025, 2025.

[38] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from well-founded assumptions. In S. Khuller and V. V. Williams, editors, *53rd ACM STOC*, pages 60–73, Virtual Event, Italy, June 21–25, 2021. ACM Press.

[39] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In O. Dunkelman and S. Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[40] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[41] A. Kiayias and M. Yung. Self protecting pirates and black-box traitor tracing. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 63–79, Santa Barbara, CA, USA, Aug. 19–23, 2001. Springer Berlin Heidelberg, Germany.

[42] V. Koppula, A. B. Lewko, and B. Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press.

[43] K. Kurosawa and Y. Desmedt. Optimum traitor tracing and asymmetric schemes. In K. Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 145–157, Espoo, Finland, May 31 – June 4, 1998. Springer Berlin Heidelberg, Germany.

[44] M. Leslie and R. Dutta. A Traceable Threshold Asmuth–Bloom Secret Sharing Scheme. Cryptology ePrint Archive, Paper 2025/1561, 2025.

[45] G. Lu, S. Nassar, and B. Waters. Succinct computational secret sharing for monotone circuits, 2025.

[46] M. Mignotte. How to share a secret? In T. Beth, editor, *EUROCRYPT'82*, volume 149 of *LNCS*, pages 371–375, Burg Feuerstein, Germany, Mar. 29 – Apr. 2, 1983. Springer Berlin Heidelberg, Germany.

[47] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62, Santa Barbara, CA, USA, Aug. 19–23, 2001. Springer Berlin Heidelberg, Germany.

[48] M. Naor and B. Pinkas. Threshold traitor tracing. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 502–517, Santa Barbara, CA, USA, Aug. 23–27, 1998. Springer Berlin Heidelberg, Germany.

[49] R. Nishimaki, D. Wichs, and M. Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 388–419, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[50] R. Safavi-Naini and Y. Wang. Sequential traitor tracing. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 316–332, Santa Barbara, CA, USA, Aug. 20–24, 2000. Springer Berlin Heidelberg, Germany.

[51] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In D. B. Shmoys, editor, *46th ACM STOC*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[52] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.

[53] V. Vinod, A. Narayanan, K. Srinathan, C. P. Rangan, and K. Kim. On the power of computational secret sharing. In T. Johansson and S. Maitra, editors, *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 162–176, New Delhi, India, Dec. 8–10, 2003. Springer Berlin Heidelberg, Germany.

[54] H. Wee. Functional encryption for quadratic functions from k-lin, revisited. In R. Pass and K. Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 210–228, Durham, NC, USA, Nov. 16–19, 2020. Springer, Cham, Switzerland.

[55] M. Zhandry. New techniques for traitor tracing: Size $N^{1/3}$ and more from pairings. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 652–682, Santa Barbara, CA, USA, Aug. 17–21, 2020. Springer, Cham, Switzerland.

[56] M. Zhandry. Optimal traitor tracing from pairings. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 305–335, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

A An Impossibility result

In this section, we prove that it is impossible to achieve strong traceability (as defined in Definition 4.7) while also achieving tracer secrecy if the secret comes from a large space or tracer non-imputability. Informally, this result means that it is impossible to efficiently trace boxes with label $\mathcal{I}$ such that $\mathcal{I} \cap \mathcal{J} \neq \emptyset$, if

we also want to achieve security against a malicious tracer. We prove this in two parts. Theorem A.1 proves that it is impossible for a TSS scheme with strong traceability to achieve tracer secrecy if the secret comes from a large space. Theorem A.4 proves that it is impossible for a TSS scheme with strong traceability to achieve tracer non-imputability.

Theorem A.1. *There does not exist any traceable secret sharing scheme for large secret spaces i.e.,* $|\mathcal{S}_\lambda| = \exp(\lambda)$*, which satisfies strong traceability and tracer secrecy for the threshold access structure.*

Proof. Let t denote the threshold and n denote the number of parties. Now, consider an adversary that samples the two secrets uniformly i.e., $s_0, s_1 \leftarrow\!\!\$\ \mathcal{S}_\lambda$, and corrupts f parties, specifically $1, \ldots, f$. It then constructs a box R^* with label $\mathcal{I} = \{f/2+1, \ldots, t\}$. When given as input shares $\mathsf{sh}'_{f/2+1}, \ldots, \mathsf{sh}'_t$, it aborts if the input shares for corrupt parties do not match the actual shares, i.e., if $\mathsf{sh}'_j \neq \mathsf{sh}_j$ for any $j \in \{f/2+1, \ldots, f\}$. Otherwise, it runs Rec with inputs pp, $\{\mathsf{sh}_i\}_{i\in[f]} \cup \{\mathsf{sh}'_i\}_{i\in\{f+1,\ldots,t\}}$, and returns its output. Informally, the box works only if the input shares of the corrupt parties match the shares it already has.

First, we claim that this box is indeed minimally-useful with respect to Definition 4.4. This is straightforward the box does indeed output the correct reconstructed secret when given real shares of all parties in $\mathcal{I}$.

Next, let e denote the min-entropy of the distribution of shares $\mathsf{sh}_{f/2+1}, \ldots, \mathsf{sh}_f$ conditioned on the tracing key tk. We now prove the theorem in two parts. Lemma A.2 proves that this entropy must be high for tracer secrecy to hold. On the other hand, Lemma A.3 proves that if this entropy is high, then the tracer cannot get the box to output anything meaningful. More formally, by Lemma A.2, if $|\mathcal{S}_\lambda| = \exp(\lambda)$, then, $1/2^e \leq \mathsf{negl}(\lambda)$ for tracer secrecy to hold. But combining this with Lemma A.3, we get that the box will output something meaningful with negligible probability, regardless of the tracing strategy. This means that the tracer cannot distinguish this box from a box that never works, and hence cannot find a traitor (unless it makes an exponential number of queries to the box).

Lemma A.2. *Let* $e, \mathcal{S}_\lambda$ *be as defined above. There exists a* PPT *adversary* $\mathcal{A}$ *such that,*

$$\mathsf{Adv}^{\mathrm{tr-sec}}_{\mathcal{A}}(\lambda) = \frac{1}{2^{e+1}} - \frac{1}{2|\mathcal{S}_\lambda - 1|} + \frac{1}{2^{e+1}|\mathcal{S}_\lambda - 1|}.$$

Proof. By the definition of e, there exists a tuple $\mathsf{sh}^*_{f/2+1}, \ldots, \mathsf{sh}^*_f$ such that the shares of parties $\{f/2+1, \ldots, f\}$ equal this tuple with probability $1/2^e$, when conditioned on the tracing key tk. The adversary $\mathcal{A}$ samples $s_0 \neq s_1 \leftarrow\!\!\$\ \mathcal{S}_\lambda$, queries shares of any $t - f/2$ parties in $[n] \setminus \{f/2+1, \ldots, f\}$. It gets the tracing key tk, the public information pp and a set of shares SH from its challenger.

We first observe that the shares of parties $\{f/2+1, \ldots, f\}$ in this game must have the same distribution (and entropy e) when conditioned on tk. This is because, we can consider the following random process which is identically distributed: the challenger first samples the tracing key, then samples the shares of parties $\{f/2+1, \ldots, f\}$ conditioned on tk, and then samples the rest of the shares.

Based on the above, $\mathcal{A}$ does the following: It simply runs the Rec algorithm with pp, SH, and $\mathsf{sh}^*_{f/2+1}, \ldots, \mathsf{sh}^*_f$ as the shares of parties $\{f/2+1, \ldots, f\}$. Let s denote the output. If $s = s_0$, it outputs 0, if $s = s_1$, it outputs 1, otherwise, it outputs a random bit i.e., $\hat{b} \leftarrow\!\!\$\ \{0,1\}$. To finish the proof, we take two cases:

- The shares of parties $\{f/2+1, \ldots, f\}$ are indeed equal to $\mathsf{sh}^*_{f/2+1}, \ldots, \mathsf{sh}^*_f$. In this case, $\mathcal{A}$ outputs the correct bit. This occurs with probability $1/2^e$.
- The shares of parties $\{f/2+1, \ldots, f\}$ are not equal to $\mathsf{sh}^*_{f/2+1}, \ldots, \mathsf{sh}^*_f$. In this case, observe that the secret value s_{1-b} was sampled uniformly at random from $\mathcal{S}_\lambda$, and the challenger behaves independently of this value. Hence, we can think of s_{1-b} as being sampled from $\mathcal{S}_\lambda \setminus \{s_b\}$ after the

challenger outputs shares SH. This means, that the probability that s_{1-b} equals the output of Rec when run on $SH \cup \{\mathsf{sh}^*_{f/2+1}, \ldots, \mathsf{sh}^*_f\}$ is bounded by $1/|\mathcal{S}_\lambda - 1|$. This means that $\mathcal{A}$ will output the wrong bit b with probability $1/|\mathcal{S}_\lambda - 1|$ in this case, and otherwise, will output the correct bit with probability at least $1/2$.

Combining the two events above, we get that the probability with which $\mathcal{A}$ guesses b correctly is $1/2^e + (1 - 1/2^e) \cdot (1 - 1/|\mathcal{S}_\lambda - 1|) \cdot 1/2$. Subtracting $1/2$ from this, we get the lemma statement. □ □

Lemma A.3. *Let R^*, e be as defined above. Then, for any* PPT *algorithm B making $Q = Q(\lambda)$ queries to R^*, it holds that,*

$$\Pr[T \neq \{\perp\}] \leq \frac{Q}{2^e},$$

where T is the set of responses of R^* for all queries done by $B(n, t, f, \mathsf{pp}, \mathsf{tk})$, and the probability is over shares $\mathsf{sh}_1, \ldots, \mathsf{sh}_t$ sampled conditioned on tk, and the random coins of B.

Proof. Let us say that the algorithm B 'wins' if the box does indeed output something other than $\perp$ in at least one of the Q queries. Now, consider a different experiment, in which, we first sample the tracing key tk, and then sample the shares of all parties conditioned on the tracing key. Moreover, the tracer always gets $\perp$ in response from R^*, but we still say that it wins if the sampled shares of parties $f/2 + 1, \ldots, f$ match the shares contained in any one of the Q queries made by the tracer to R^*. The probability that the tracer wins in this modified experiment is the same as in the original experiment since its view is identical in both experiments as long as it has not yet won. In this modified experiment, the queries of the tracer only depend on the tracing key, and otherwise are independent of the actual shares $\mathsf{sh}_{f/2+1}, \ldots, \mathsf{sh}_f$ of parties $f/2 + 1, \ldots, f$. So, we can think of shares $\mathsf{sh}_{f/2+1}, \ldots, \mathsf{sh}_f$ as being chosen after the queries are determined, conditioned on tk. In this case, the probability that the shares $\mathsf{sh}_{f/2+1}, \ldots, \mathsf{sh}_f$ do indeed match one of the Q queries of the tracer is bounded by $Q/2^e$. This proves the lemma because the box will only work in the real experiment, i.e., output something other than $\perp$, if the tracer correctly guesses these shares. □

□

Theorem A.4. *There does not exist any traceable secret sharing scheme which satisfies strong traceability and tracer non-imputability for the threshold access structure.*

of Thm. A.4. We follow a similar approach to the proof of Theorem A.1. Specifically, we focus on the threshold access structure, i.e., t-out-of-n. Consider an adversary that samples two secrets uniformly i.e., $s_0, s_1 \leftarrow\!\!\$\ \mathcal{S}_\lambda$ and corrupts parties $1, \ldots, t-1$. It then constructs a box R^* with label $\mathcal{I} = \{t-1, t\}$. When given as input shares $\mathsf{sh}'_{t-1}, \mathsf{sh}'_t$, it aborts if the share for corrupt party $t-1$ does not match the actual share, i.e., if $\mathsf{sh}'_{t-1} \neq \mathsf{sh}_{t-1}$. Otherwise, it runs Rec with inputs pp, $\{\mathsf{sh}_i\}_{i \in [t-1]} \cup \{\mathsf{sh}'_t\}$ and returns its output.

As in the proof of Theorem A.1, this box is indeed minimally-useful as per Definition 4.4.

Next, let e denote the min-entropy of share sh_{t-1} conditioned on the tracing key tk. By Lemma A.3, we know that a PPT tracer can trace the box only if it can get it to output something meaningful, which happens only if $1/2^e$ is non-negligible. Informally, the entropy of share sh_{t-1} is low conditioned on the tracing key. But now, we will prove that if this is the case, we can break tracer non-imputability. Lemma A.5 below formally proves this.

Lemma A.5. *Let e denote the log of entropy of share sh_{t-1} conditioned on the tracing key. Then, there exists* PPT *adversaries $\mathcal{A}$ and $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathrm{ni}}_{\mathcal{A}}(\lambda) \geq \frac{1}{2^e} \cdot (1 - \mathsf{Adv}^{\mathrm{str-trac}}_{\mathcal{B}}(\lambda))$$

Proof. By definition of e, there exists a value sh^*_{t-1} such that the share of party t equals this tuple with probability $1/2^e$, conditioned on the tracing key. We now construct $\mathcal{A}$ as follows: $\mathcal{A}$ samples a secret $s \leftarrow\!\!\$\ \mathcal{S}_\lambda$, and responds to its tracer non-imputability challenger with t-out-of-n threshold access policy, $i^* = t-1$, s. It will then get back the tracing key tk, verification key vk, and shares of all parties except $t-1$ i.e., $\{\mathsf{sh}_i\}_{i\in[n]\setminus\{t-1\}}$.

$\mathcal{A}$ then constructs a reconstruction box $\hat{R}$ containing the share sh^*_{t-1}, with label $\hat{I} = [t-2]\cup\{t\}$. This box simply runs the Rec algorithm with sh^*_{t-1} and the input shares, and returns the output.

Next, $\mathcal{A}$ runs the Trace algorithm on this box. Let us use $\mathcal{J}^*, \pi^*$ to denote the output. $\mathcal{A}$ simply returns $\mathcal{J}^*, \pi^*$ to its tracer non-imputability challenger.

Let us now analyze the advantage of $\mathcal{A}$. Let E_g denote the event that $\mathsf{sh}_{t-1} = \mathsf{sh}^*_{t-1}$, i.e., $\mathcal{A}$ correctly guesses the share of party $t-1$. Observe that this event occurs with probability $1/2^e$. Moreover, conditioned on this event, $\hat{R}$ is indeed a minimally-useful box.

It is easy to see that, conditioned on E_g, if $\mathcal{J}^* = \emptyset$ or if $\mathcal{J}^* \neq \{t-1\}$ or if $\mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi^*) = 0$, we can use $\mathcal{A}$ to break strong traceability. More formally, we can construct a PPT adversary $\mathcal{B}$ that sends the t-out-of-n access structure, along with $\mathcal{J} = \{t-1\}$, and randomly sampled secrets s_0, s_1 to its challenger. On receiving pp and sh_{t-1} from the challenger, $\mathcal{B}$ constructs a box using sh_{t-1} exactly as $\mathcal{A}$ constructs $\hat{R}$ above. Clearly, this is a minimally-useful box. Moreover, both $\mathcal{J}$ and $\mathcal{I}$ are unauthorized. Hence, $\mathcal{B}$ wins the traceability game if $\mathcal{J}^* = \emptyset$ or if $\mathcal{J}^* \neq \{t-1\}$ or if $\mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi^*) = 0$.

Lastly, observe that if $\mathcal{J}^* = \{t-1\}$ and $\mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi^*) = 1$, then $\mathcal{A}$ wins its game. This proves the theorem. □

□

B GTSS with Strong Traceability using iO

In this section, we extend our iO-based construction from Section 5 to achieve strong traceability. Note that we do not achieve tracer secrecy or tracer non-imputability in this case because of the impossibility result presented in Appendix A.

Correctness. Correctness follows directly from correctness of the doubly constrained signature and obfuscation. Specifically, for any set $J \subseteq [n]$ such that $C(x_J) = 1$, when the program $\hat{P}$ is run with inputs $\{\mathsf{sh}_i\}_{i\in J}$, the signatures in all the shares would verify by correctness of DCS. Moreover, since the ciphertext in each share decrypts to $(0,0)$, the resulting set $\hat{J}$ will simply equal J. Lastly, since $C(x_J) = 1$, the program would indeed output the correct secret s, by correctness of Obf.

Secrecy. Theorem B.1 below proves secrecy of our scheme based on the security of DCS and Obf.

Theorem B.1. *For all* PPT *adversaries* D*, there exist adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *such that,*

$$\mathsf{Adv}^{\mathsf{sec}}_{D,\mathsf{iOsGTSS}}(\lambda) \leq \mathsf{Adv}^{\mathsf{sec}}_{\mathcal{B}_1,\mathsf{DCS}}(\lambda) + \mathsf{Adv}^{\mathsf{sec}}_{\mathcal{B}_2,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathsf{sec}}_{\mathcal{B}_3,\mathsf{Obf}}(\lambda).$$

Proof. We prove the theorem via a series of hybrids. Let us denote the original game by H_0, with s_0, s_1 as the two secrets, and $\mathcal{J}$ as the unauthorized set corrupted by $\mathcal{A}$.

We now define a hybrid H_1, where we generate a constrained key pair for DCS. Specifically, instead of running KeyGen, the challenger runs $\mathsf{ConstPK}$, with $C_1 = 1$ and circuit C_2 which outputs 1 on input (i, ct_i) if and only if $i \in \mathcal{J}$. In other words, there are no valid signatures on any party outside of the corrupt set $\mathcal{J}$. Note that, wile C_1 is not a subset of C_2, it does not affect our proof since the adversary does not see the constrained secret key in this game.

We claim that if an adversary can distinguish between H_0 and H_1, it can be used to break security of the DCS scheme. To see why, observe that the adversary only sees signatures which are included in shares of the corrupt parties $\mathcal{J}$, i.e., signatures on messages which satisfy the circuit C_2.

$\underline{\mathsf{Share}(1^\lambda, s, C)}$:

1. Sample PRF key $k \leftarrow\!\!\$ \{0,1\}^\lambda$ and doubly-constrained signature keys $(\mathsf{pk}, \mathsf{sk}, \mathsf{sk}_1) \leftarrow\!\!\$ \mathsf{DCS.KeyGen}(1^\lambda, \ell, z, C_1 = 1)$. / $\ell = \log(n) + \lambda + 1 + \log(n+1)$, $\mathsf{CT} := \{0,1\}^\lambda \times \{0,1\}^{1+\log(n+1)}$ and $z = z(\lambda, n)$ will be specified later.
2. For all $i \in [n]$, compute $\mathsf{ct}_i := \mathsf{Enc}(k, (0,0))$ and $\sigma_i := \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}_i))$. Set $\mathsf{sh}_i \leftarrow (i, \mathsf{ct}_i, \sigma_i)$.
3. Set $\mathsf{tk} \leftarrow (k, \mathsf{sk}, \{\mathsf{sh}_i\}_{i\in[n]})$.
4. Obfuscate the program P described below, i.e., $\hat{\mathsf{P}} \leftarrow \mathsf{Obf}(\mathsf{P})$.
5. Output $\{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{pp} \leftarrow \hat{\mathsf{P}}, \mathsf{tk}, \mathsf{vk} \leftarrow \bot$.

$\underline{\mathsf{Enc}(k, (f, j))}$:

1. Sample $r \leftarrow\!\!\$ \{0,1\}^\lambda$.
2. Compute $\mathsf{ct}' \leftarrow \mathsf{PRF.F}(k, r) + (f||j)$.
3. Output $\mathsf{ct} \leftarrow (r, \mathsf{ct}') \in \mathsf{CT}$.

$\underline{\mathsf{P}(\{i, \mathsf{ct}_i, \sigma_i\}_{i\in J})}$:

1. **Constants**: C, k, pk, s.
2. Parse ct_i as (r, ct'_i).
3. Let $\hat{J} = J$. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
4. For all $i \in \hat{J}$, let $(f_i, j_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
5. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1, \ldots, n\}$. Compute $S \leftarrow \bigcap_{i\in\hat{J}} S_i$.
6. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
7. Set $\hat{J} = \hat{J} \cap S$.
8. Abort if $C(x_{\hat{J}}) = 0$.
9. Otherwise, output s.

$\underline{\mathsf{Rec}(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in J})}$:

1. Parse pp as $\hat{\mathsf{P}}$.
2. Output $\hat{\mathsf{P}}(\{\mathsf{sh}_i\}_{i\in J})$.

Figure 23: The Share and Rec algorithms for our iO-based Traceable secret sharing scheme iOsGTSS with strong traceability. The Dec algorithm is as described in Fig. 8.

$\underline{\mathsf{Trace}^R(\mathsf{pp}, \mathsf{tk}, 1^{1/\epsilon}, s_0, s_1, \mathcal{I}):}$

1. Parse tk as $(k, \mathsf{sk}, \{\mathsf{sh}_i\}_{i\in[n]})$ and pp as $\hat{\mathsf{P}}$. Let $\mathcal{J}^* := \bot$.
2. Run $\hat{\mathsf{P}}$ on inputs $\{\mathsf{sh}_i\}_{i\in J}$ for any $J \subseteq [n]$ such that $C(x_J) = 1$. Let the output be s.
3. Let $\mathcal{I} = \{i_1, \ldots, i_f\}$. For $j \in \{0, \ldots, f\}$:
   (a) Let $q_j := 0$. For $q \in \{1, \ldots, N\}$: // $N = \frac{8\lambda \cdot n^2}{\epsilon^2}$
      i. For $i \in \{i_1, \ldots, i_j\}$: compute $\mathsf{ct}''_i \leftarrow \mathsf{Enc}(k, (0,0))$ and $\sigma''_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}''_i))$. Let $\mathsf{sh}''_{i,j} \leftarrow (i, \mathsf{ct}''_i, \sigma''_i)$.
      ii. For $i \in \{i_{j+1}, \ldots, i_f\}$, set $\mathsf{sh}''_{i,j} \leftarrow \mathsf{sh}_i$.
      iii. Query R on $\{\mathsf{sh}''_{i,j}\}_{i\in\mathcal{I}}$. Let b denote R's response.
      iv. If $s_b = s$, then set $q_j \leftarrow q_j + 1$.
   (b) Compute $q'_j \leftarrow q_j/N$. If $j > 0$ and $q'_{j-1} - q'_j > \epsilon/4n$, then set $\mathcal{J}^* \leftarrow \mathcal{J}^* \cup \{i_j\}$.
4. If $\mathcal{J}^* \neq \emptyset$, output $\mathcal{J}^*$.
5. Compute $\hat{\epsilon} = \epsilon/2$. For $j \in \{0, \ldots, n\}$:
   (a) Let $p_j \leftarrow 0$. For $q \in \{1, \ldots, \hat{N}\}$: // $\hat{N} = \frac{8\lambda \cdot n^2}{\hat{\epsilon}^2}$
      i. For all $i \in \mathcal{I}$, compute $\mathsf{ct}'_i \leftarrow \mathsf{Enc}(k, (1, j))$ and $\sigma'_i \leftarrow \mathsf{DCS.Sign}(\mathsf{sk}, (i, \mathsf{ct}'_i))$.
      ii. Query R on $\{\mathsf{sh}^*_{i,j} \leftarrow (i, \mathsf{ct}'_i, \sigma'_i)\}_{i\in\mathcal{I}}$. Let b denote R's response.
      iii. If $s_b = s$, then set $p_j \leftarrow p_j + 1$.
   (b) Compute $p'_j \leftarrow p_j/\hat{N}$. If $j > 0$ and $p_{j-1} - p_j > \hat{\epsilon}/4n$, then set $\mathcal{J}^* \leftarrow \mathcal{J}^* \cup \{j\}$.
6. Output $\mathcal{J}^*$.

Figure 24: The Trace algorithm for our iO-based Traceable secret sharing scheme iOsGTSS with strong traceability.

More formally, we will construct an adversary D_1 that breaks security of DCS from D that can distinguish between the two hybrids. D_1 acts as the challenger to D. It samples $b_1, b_1^* \leftarrow\$ \{0,1\}$, and sets $s \leftarrow s_{b_1}$ and $s^* \leftarrow s_{b_1^*}$.

D_1 then samples a PRF key k, and gives as input ℓ, z, C_1 and C_2 as defined above, to its challenger. It then gets pk from its challenger and computes shares of all parties $i \in \mathcal{J}$. For each share, it computes the ciphertext $\mathsf{ct}_i \leftarrow\$ \mathsf{Enc}(k, (0,0))$, and then queries the SignO oracle of its challenger to get a signature on (i, ct_i). It obfuscates the program as given in Fig. 23. D_1 then sends $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$ and pp to its adversary. It then forwards D's response to its challenger. It is clear that D_1 perfectly simulates the game to D, and hybrid H_0 corresponds to $b = 0$ in the DCS security game, and H_1 corresponds to $b = 1$. Hence, we get that,

$$|\mathsf{Adv}^{\mathrm{sec}}_{D,\mathsf{iOGTSS},H_0}(\lambda) - \mathsf{Adv}^{\mathrm{sec}}_{D,\mathsf{iOGTSS},H_0}(\lambda)| = \mathsf{Adv}^{\mathrm{sec}}_{D_1,\mathsf{DCS}}(\lambda).$$

H_2: We hard-code $\mathcal{J}$ in P, and directly set $\hat{J}$ to be $J \cap \mathcal{J}$. We argue that the functionality of the program remains the same. To see that, observe that if J contains parties outside of the set $\mathcal{J}$, they will not pass the verification check DCS.SignVerify, because there is no valid signature for these parties as per the constrained public key. Hence, these parties would have been removed from $\hat{J}$ anyway. Hence, by security of Obf, H_2 looks indistinguishable from H_1.

H_3: We again modify the program P. Specifically, we remove the secret s from the program, and simply remove the last line. We argue that the functionality does not change. To see that, observe that the resulting set $\hat{J}$ will be a subset of $\mathcal{J}$ regardless of the input. Moreover, since $C(x_{\mathcal{J}}) = 0$, this means $C(x_{\hat{J}}) = 0$. Hence, the program would never reach the last line. This proves that by security of Obf, H_3 looks indistinguishable from H_2.

Since the view of the adversary in hybrid H_3 contains no information about the bit b, the adversary's advantage in guessing b is bounded by $1/2$. This proves the theorem. □ □

Strong Traceability. Theorem B.2 below proves strong traceability for our scheme. We rely on security of DCS, Obf and the punctured PRF′. Note that we prove a less-adaptive notion of strong traceability, wherein the adversary sends the label $\mathcal{I}$ along with the corrupt set $\mathcal{J}$ in the first step of the game. We refer to this game as semi-adaptive strong tracing $\mathbf{ExpStrongTrace}^{sa}$, and define advantage of an adversary $\mathcal{A}$ as:

$$\mathsf{Adv}^{\mathrm{sa-str-trac}}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) := \Pr[\mathbf{ExpStrongTrace}^{sa}_{\mathcal{A},\mathsf{GTSS},\epsilon} = 1].$$

Theorem B.2. *If* DCS, Obf *and the puncturable* PRF *schemes are secure, then, for all* PPT *adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathrm{sa-str-trac}}_{\mathcal{A},\mathsf{GTSS},\epsilon}(\lambda) \leq \mathsf{negl}(\lambda).$$

Proof. We prove the theorem in two parts. First, we prove that the tracer will never blame an honest party in step 3. Second, we consider the case where the tracer finds no party in step 3. Here, we prove that the tracer must find at least one party in step 5 if the event GoodBox occurs, and this party can never be honest.

We start with some notation. Let us use s_0, s_1 to denote the secrets given by $\mathcal{A}$. Let $s = s_b$ denote the secret that is shared in the Share algorithm, where b is sampled uniformly at random from $\{0,1\}$. Let $\mathcal{I}$ denote the label of the reconstruction box R, and let $\mathcal{J}$ denote the set of corrupt parties. Let E_1 denote the event that the tracer finds no corrupt party in step 3.

For the first part, let $\hat{q}_j$ denote the probability that the box R outputs the correct bit b when given shares $\{\mathsf{sh}''_{i,j}\}_{i\in\mathcal{I}}$ as defined in Fig. 24, for any $j \in \{0,\ldots,f\}$. Lemma B.3 below then proves that the tracer will never blame an honest party in step 3, except with negligible probability.

Lemma B.3. *For all $j \in [f]$ such that $i_j \in \mathcal{I} \setminus \mathcal{J}$,*

$$q'_{j-1} - q'_j \leq \epsilon/8n,$$

except with negligible probability.

Proof. For any honest party $i_j \in \mathcal{I} \setminus \mathcal{J}$, observe that the view of the adversary contains no information about the share of this party. Hence, the following distributions are identical, from the adversary's view:

$$\{\mathsf{sh}''_{i_1,j-1}, \ldots, \mathsf{sh}''_{i_{j-1},j-1}, \mathsf{sh}_{i_j}, \ldots, \mathsf{sh}_{i_f}\} \approx_s \{\mathsf{sh}''_{i_1,j}, \ldots, \mathsf{sh}''_{i_j,j}, \mathsf{sh}_{i,j+1} \ldots, \mathsf{sh}_{i_f}\}$$

This is because the share $\mathsf{sh}''_{i,j}$ is sampled with the same random process as $\mathsf{sh}_{i,j}$, i.e., by sampling an encryption to $(0,0)$ and then generating a signature. Moreover, the shares $\mathsf{sh}''_{i,j-1}$ and $\mathsf{sh}''_{i,j}$ are identically distributed for all $i \in \{i_1, \ldots, i_{j-1}\}$. Hence, $\hat{q}_{j-1} = \hat{q}_j$.

Next, by Chernoff bound, we know that, for all $j \in \{0, \ldots, f\}$, $|\hat{q}_j - q'_j| \leq \epsilon/16n$, with probability $1 - 2e^{-\lambda/16}$. Hence, we get:

$$\begin{aligned} q'_{j-1} - q'_j &= q'_{j-1} - \hat{q}_{j-1} + \hat{q}_{j-1} - \hat{q}_j + \hat{q}_j - q'_j && (8)\\ &\leq \epsilon/16n + 0 + \epsilon/16n. && (9) \end{aligned}$$

This proves the lemma. □

We now move on to the second part, i.e., event E_1 occurs wherein no party was found in step 3. By definition of a minimally-useful box, we know that, $\hat{q}_0 = 1/2 + \epsilon$, since the inputs $\{\mathsf{sh}_i\}_{i\in\mathcal{I}}$ are valid shares. Moreover, since no party is found in step 3, this implies $q'_{j-1} - q'_j \leq \epsilon/4n$ for all $j \in [f]$. Adding this equation for all $j \in [f]$, we get $q'_0 - q'_f \leq (\epsilon/4n) \cdot |\mathcal{I}| \leq \epsilon/4$. Moreover, by Chernoff bound, we know for all $j \in \{0, \ldots, f\}$, $|\hat{q}_j - q'_j| \leq \epsilon/16n$, except with negligible probability. Combining all these together, we get,

$$\begin{aligned} \hat{q}_f &\geq q'_f - \epsilon/16n && (10)\\ &\geq q'_0 - \epsilon/4 - \epsilon/16n && (11)\\ &\geq \hat{q}_0 - \epsilon/16n - \epsilon/4 - \epsilon/16n && (12)\\ &\geq 1/2 + \epsilon - \epsilon/16n - \epsilon/4 - \epsilon/16n && (13)\\ &\geq 1/2 + \epsilon/2 && (14) \end{aligned}$$

In other words, the box gives the correct output b with probability $1/2 + \hat{\epsilon}$ when given as input freshly sampled shares for parties in $\mathcal{I}$, except with negligible probability.

We will now prove that the tracer will indeed blame at least one party in step 5 in the event E_1. We will finish the proof by proving that the blamed party cannot be honest.

Lemma B.4 below proves that if $\hat{q}_f \geq 1/2 + \hat{\epsilon}$, then, the box R must output b with roughly the same probability, when given shares encrypting $(1,0)$ for all parties in $\mathcal{I}$.

Lemma B.4. *If the events E_1 and $\mathsf{GoodBox}$ occur and $\mathsf{DCS}, \mathsf{Obf}$ and the puncturable PRF' schemes are secure, then*

$$\Pr[R(\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}) = b] \geq \frac{1}{2} + \hat{\epsilon} - \mathsf{negl}(\lambda),$$

where $\{\mathsf{sh}^*_{i,0}\}_{i\in\mathcal{I}}$ are as defined in line 5(a)ii of the Trace algorithm.

Proof. As proven above, in the event E_1, we have $\hat{q}_f \geq 1/2 + \hat{\epsilon}$. Let H_0 refer to this original game, wherein the challenger runs R with shares $\{\mathsf{sh}''_{i,f}\}_{i \in \mathcal{I}'}$. For all $i \in \mathcal{J}$, let $\mathsf{sh}_i = (i, \mathsf{ct}_i, \sigma_i)$, and for all $i \in \mathcal{I}$, let $\mathsf{sh}''_{i,f} = (i, \mathsf{ct}'_i, \sigma'_i)$ be the shares sampled by the challenger. We will now define a series of hybrids.

H_1: We change how we sample the keys of the DCS scheme. Specifically, the challenger first samples the ciphertexts for all parties in $\mathcal{J} \cup \mathcal{I}$. Then, instead of running KeyGen as in the honest Share algorithm, the challenger samples a constrained key-pair, by running ConstPK. The input circuit C_1 is just 1, as in the Share algorithm, and the circuit C_2 is defined as follows: $C_2(i, \mathsf{ct}''_i)$ is 1 if and only if, (i) $i \in \mathcal{J} \cup \mathcal{I}$ and (ii) $\mathsf{ct}''_i = \mathsf{ct}_i$ for $i \in \mathcal{J} \setminus \mathcal{I}$, $\mathsf{ct}''_i = \mathsf{ct}'_i$ for $i \in \mathcal{I} \setminus \mathcal{J}$ and $\mathsf{ct}''_i \in \{\mathsf{ct}_i, \mathsf{ct}'_i\}$ for $i \in \mathcal{J} \cap \mathcal{I}$, i.e., the ciphertext must match the ones sampled by the challenger. Note that C_1 is not a subset of C_2 as defined, but that does not affect our security proof since the adversary in this game never sees the tracing key.

We claim that if the box's behavior differs by a non-negligible amount from H_0 to H_1, we can use it to break security of the DCS scheme. To see why, observe that the view of the adversary only includes the obfuscated program which contains the public key pk, and honestly-generated shares of $\mathcal{J} \cup \mathcal{I}$ – each of these shares contains a signature on messages which satisfy C_2. Hence, we can make an adversary $\mathcal{B}_1$ that acts as the challenger to $\mathcal{A}$, and uses it to win the DCS security game.

Specifically, $\mathcal{B}_1$ gets as input $(C, \mathcal{J}, \mathcal{I}, s_0, s_1)$ from $\mathcal{A}$ (it gets $\mathcal{I}$ because we're considering the semi-adaptive tracing game). It samples a PRF key k, and computes ciphertexts encrypting $(0, 0)$ for shares of all parties in $\mathcal{J} \cup \mathcal{I}$, i.e., $\mathsf{ct}_i \; \forall \; i \in \mathcal{J}$ and $\mathsf{ct}'_i \; \forall \; i \in \mathcal{I}$.

It then forwards to its challenger (ℓ, z, C_1, C_2), wherein $C_1 = 1$, and C_2 is as defined above, and gets back a public key pk. To generate shares sh_i for $\mathcal{J}$, it queries the SignO oracle to get a signature on (i, ct_i). It samples b randomly from $\{0, 1\}$, obfuscates program P containing pk, s_b, as defined in Fig. 23, and sends to $\mathcal{A}$ the obfuscated program along with $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$.

It then gets a box R from $\mathcal{A}$. It generates shares $\mathsf{sh}''_{i,f}$ for all $i \in \mathcal{I}$, by querying the SignO oracle to get signatures on (i, ct'_i). It runs R with the shares $\{\mathsf{sh}''_{i,f}\}_{i \in \mathcal{I}}$. If R outputs b then it sends 0 to its challenger, otherwise it sends 1. Clearly, $\mathcal{B}_1$ perfectly simulates the two hybrids to $\mathcal{A}$ – H_0 (and H_1) corresponds to $b = 0$ (and $b = 1$) in the DCS security game. Hence, we get that R must still output b with probability $\geq \frac{1}{2} + \hat{\epsilon} - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1, \mathsf{DCS}}(\lambda)$ in this hybrid.

H_2: We now change the program P as follows. We hard-code the sets $\mathcal{J}$ and $\mathcal{I}$ in the program, along with the shares $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$ and $\{\mathsf{sh}''_{i,f}\}_{i \in \mathcal{I}}$. Moreover, before checking any signatures, we first set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$, and for input shares of parties in $(\mathcal{J} \cup \mathcal{I})$, we remove them from $\hat{J}$ if their input does not match the hard-coded share. Figure 25 shows the full program.

We argue that this change does not alter the functionality of the program. To see that, observe that there are no valid signatures on parties outside of $(\mathcal{J} \cup \mathcal{I})$, so these parties would have been excluded from $\hat{J}$ anyways. Similarly, for parties in $(\mathcal{J} \cup \mathcal{I})$, there exist valid signatures only for the shares hard-coded in the box. Hence, the resulting $\hat{J}$ is the same for any input across the two hybrids. More formally, this means that if the box behaves differently in hybrids H_1 and H_2, we can use $\mathcal{A}$ to break security of obfuscation. Hence, R must output b in this hybrid with probability $\geq \frac{1}{2} + \hat{\epsilon} - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1, \mathsf{DCS}}(\lambda) - \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_2, \mathsf{Obf}}(\lambda)$.

H_3: In hybrid H_3, we again change the program P. We now puncture the PRF key k. Let $\mathsf{sh}''_{i,f} = (i, (r'_i, \hat{\mathsf{ct}}'_i), \sigma'_i)$ for all $i \in \mathcal{I}$. We puncture k at the set $T := \{r'_i\}_{i \in \mathcal{I}}$, and hard-code $z_r := \mathsf{PRF.F}(k, r)$ in the program for all $r \in T$. To decrypt the ciphertexts, we now use these hard-coded values. H_3 should be indistinguishable from H_2 to $\mathcal{A}$ by security of obfuscation, since the functionality is identical.

H_4: We can now replace each z_r with a uniformly random value. This is indistinguishable from the previous hybrid by the security of punctured PRFs. This now means that by one-time pad security, if the r_i used in all ciphertexts for $\mathcal{J} \cup \mathcal{I}$ are unique, then, each ciphertext $\{\hat{\mathsf{ct}}'_i\}_{i \in \mathcal{I}}$ is now uniformly random, except with probability $\approx n^2/2^\lambda$.

H_5: We can now change the ciphertexts in shares of parties $\mathcal{I}$ to encrypt $(1, 0)$ instead of $(0, 0)$ by simply adding $(1, 0)$ to each z_r value. We argue that this change does not affect the functionality. Specifically,

$\underline{\mathsf{P}(\{i, \mathsf{ct}_i, \sigma_i\}_{i \in J})}$:

1. **Constants**: C, k, pk, s and $\mathcal{J}, \mathcal{I}$, $\{\mathsf{sh}_i\}_{i \in \mathcal{J}}$ and $\{\mathsf{sh}''_{i,f}\}_{i \in \mathcal{I}}$.
2. Parse ct_i as (r, ct'_i).
3. Let $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$.
4. For all $i \in (\mathcal{J} \setminus \mathcal{I}) \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
5. For all $i \in (\mathcal{I} \setminus \mathcal{J}) \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}''_{i,f}$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
6. For all $i \in (\mathcal{I} \cap \mathcal{J}) \cap J$, if $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}''_{i,f}$ and $(i, \mathsf{ct}_i, \sigma_i) \neq \mathsf{sh}_i$ then set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
7. For all $i \in J$, if $\mathsf{DCS.SignVerify}(\mathsf{pk}, (i, \mathsf{ct}_i), \sigma_i) = 0$, set $\hat{J} \leftarrow \hat{J} \setminus \{i\}$.
8. For all $i \in \hat{J}$, let $(f_i, j_i) \leftarrow \mathsf{Dec}(k, \mathsf{ct}_i)$.
9. For all $i \in \hat{J}$, compute $S_i \leftarrow \{j_i + 1, \ldots, n\}$. Compute $S \leftarrow \bigcap_{i \in \hat{J}} S_i$.
10. For all $i \in \hat{J}$ such that $f_i = 1$, set $S \leftarrow S \cup \{i\}$.
11. Set $\hat{J} = \hat{J} \cap S$.
12. Abort if $C(x_{\hat{J}}) = 0$.
13. Output s.

Figure 25: The modified program used in hybrid H_2 in the strong traceability proof for our scheme iOsGTSS.

for any input set, we know that $\hat{J}$ is a subset of $\mathcal{J} \cup \mathcal{I}$, with ciphertexts equal to the hard-coded shares for these parties. Since all these ciphertexts encrypt $(0,0)$ in H_4, the set $S_i = [n]$ for all parties hence $S = [n]$. Similarly, in H_5, the ciphertexts for parties in $\mathcal{J}$ and $\mathcal{I}$ respectively encrypt $(0,0)$ and $(1,0)$. In either case, $S_i = [n]$ for all i, and hence, $S = [n]$. So the resulting set $\hat{J}$ will be the same across the two hybrids, for all inputs.

This means that the box must output b with probability $1/2 + \hat{\epsilon} - \mathsf{negl}(\lambda)$ even when run with shares $\mathsf{sh}^*_{i,0}$ for all $i \in \mathcal{I}$. □

□

Next, Lemma B.5 below proves that the box cannot output b with probability more than $1/2$ when given shares with ciphertexts encrypting $(1, n)$.

Lemma B.5. *If the event* GoodBox *occurs then,*

$$\Pr[R(\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}) = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{sh}^*_{i,n}$ *shares are as defined in line 5(a)ii of the* Trace *algorithm.*

Proof. We again make a series of hybrids. Let H_0 denote the original strong traceability game, wherein the box R is given as inputs $\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{I}}$. Let $\mathsf{sh}^*_{i,n} = (i, \mathsf{ct}^*_{i,n}, \sigma^*_{i,n})$ for all $i \in \mathcal{I}$.

We define a new hybrid H_1, where we change how we sample the keys of the doubly-constrained signature scheme DCS. First, the challenger samples ciphertexts for parties in $\mathcal{J} \cup \mathcal{I}$, by encrypting $(0,0)$ for parties in $\mathcal{J}$ and $(1,n)$ for parties in $\mathcal{I}$. Then, instead of running KeyGen as in the honest Share algorithm, the challenger runs ConstPK to generate a constrained key-pair. The circuit $C_1 = 1$ is as defined in the Share algorithm, and C_2 is constructed in the same way as in hybrid H_1 in the proof of Lemma B.4. $C_2(i, \mathsf{ct}''_i)$ is 1 if and only if (i) $i \in \mathcal{J} \cup \mathcal{I}$ and (ii) the ciphertext matches the ones sampled by the challenger. Note that C_1 is not a subset of C_2 but that does not affect this proof since the adversary never sees the constrained signing key sk_{C_1}.

Similar to hybrid H_1 in the proof of Lemma B.4, if the box behaves differently in H_1 than in H_0, we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks security of the DCS scheme.

Next, in hybrid H_2, we change the program P, similar to hybrid H_2 in proof of Lemma B.4. We hard-code the sets $\mathcal{J}$ and $\mathcal{I}$ and their corresponding shares $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$, $\{\mathsf{sh}^*_{i,n}\}_{i\in\mathcal{J}}$. Moreover, before checking any signatures, we first set $\hat{J} = J \cap (\mathcal{J} \cup \mathcal{I})$, and for input shares of parties in $(\mathcal{J} \cup \mathcal{I})$, we remove them from $\hat{J}$ if their input does not match the hard-coded share. The full program is similar to Figure 25, with shares $\mathsf{sh}''_{i,f}$ replaced with $\mathsf{sh}^*_{i,n}$. By an argument similar to that in Lemma B.4, the functionality of the program is unchanged, and so the box must behave similarly in hybrids H_1 and H_2.

In hybrid H_3, we modify the program again – we simply remove the last line, and remove s from the program. We argue that this does not change the functionality of the program. To see that, observe that the resulting set $\hat{J}$ in the program in H_2 is always a subset of $\mathcal{J} \cup \mathcal{I}$. Moreover, if $\hat{J}$ is a subset of $\mathcal{J}$, then the program would abort anyway since $C(x_{\mathcal{J}}) = 0$. So we only need to worry about inputs such that the resulting $\hat{J}$ has at least one share from $\mathcal{I} \setminus \mathcal{J}$. But for any party in $\mathcal{I} \setminus \mathcal{J}$, the only valid share is $\mathsf{sh}^*_{i,n} = (i, \mathsf{ct}^*_{i,n}, \sigma^*_{i,n})$ (since there is no valid signature on any other ciphertext), where $\mathsf{ct}^*_{i,n}$ decrypts to $(1,n)$. This means that $S_i = \emptyset$, and we will only include i in the set S. Hence the set S will end up being a subset of $\mathcal{I}$, resulting in $\hat{J} \subseteq \mathcal{I}$. But since $C(x_{\mathcal{I}}) = 0$, the program will always abort for these inputs as well. Hence, it never reaches the last line, and so the functionality of the modified program is the same as that in H_2.

But now observe that in hybrid H_3, the view of the adversary contains no information about b, and hence, the probability of R outputting b correctly cannot be more than $1/2$. This proves the lemma. □ □

Let $\hat{p}_j$ denote the probability with which the box R outputs b when run with shares $\{\mathsf{sh}^*_{i,j}\}_{i\in\mathcal{I}}$. Lemma B.4 proves that $\hat{p}_0 = \frac{1}{2} + \hat{\epsilon} - \mathsf{negl}(\lambda)$. Lemma B.5 proves that $\hat{p}_n \leq \frac{1}{2} + \mathsf{negl}(\lambda)$. Combining the two, we get that there must be a jump in these probabilities for some $j \in [n]$. Formally, there exists at least one $j^* \in [n]$ such that, $\hat{p}_{j^*-1} - \hat{p}_{j^*} \geq \hat{\epsilon}/2n$.

Next, observe that the values p'_j computed in the Trace algorithm are an approximation of the probability $\hat{p}_j$. By Chernoff bound, for all $j \in \{0, \ldots, n\}$,

$$\Pr[|\hat{p}_j - p'_j| > \frac{\hat{\epsilon}}{16n}] < 2e^{-\lambda/64}.$$

Hence, we get that, for j^*, $p'_{j^*-1} - p'_{j^*} \geq \hat{\epsilon}/2n - \hat{\epsilon}/8n > \hat{\epsilon}/4n$. This proves that if the events $\mathsf{GoodBox}$ and E_1 occur, then the Trace algorithm will blame at least one party i.e., j^*. Hence, $\mathcal{J}^* \neq \perp$, except with negligible probability in the event $\mathsf{GoodBox} \wedge \mathsf{E}_1$.

We now move to the last part, where we prove that the jump found by the tracer can not be honest. We prove this via two lemmas.

Lemma B.6. *Let $\mathcal{J}, \mathcal{I}$ and $\{\hat{p}_j\}_{j\in\{0,1,\ldots,n\}}$ be as defined above. For all $j \in [n] \setminus (\mathcal{J} \cup \mathcal{I})$,*

$$|\hat{p}_j - \hat{p}_{j-1}| \leq \mathsf{negl}(\lambda).$$

Proof. We make a series of hybrids to prove this. Let H_0 denote the original game where the challenger honestly runs Share, and runs the box R with $\{\mathsf{sh}^*_{i,j-1}\}_{i\in\mathcal{I}}$.

We define H_1, wherein the challenger samples a constrained key-pair for the DCS scheme. Specifically, it first samples the PRF key k, and ciphertexts for shares of parties in $\mathcal{J}$ i.e., ct_i encrypting $(0, 0)$ and ciphertexts $\mathsf{ct}^*_{i,j-1}$ for parties in $\mathcal{I}$ encrypting $(1, j-1)$. It then runs ConstPK instead of KeyGen, with inputs ℓ, z and $C_1 = 1$ as in the Share algorithm, and C_2 defined as follows: $C_2(i, \mathsf{ct}''_i)$ outputs 1 if, (i) $i \in \mathcal{J} \cup \mathcal{I}$ and ct''_i matches the ciphertexts sampled by the challenger for i, or (ii) $i \neq j$. In other words, there are no valid signatures for party j with respect to the constrained key.

We claim that if the adversary can distinguish between hybrids H_0 and H_1, we can use it to make an adversary $\mathcal{B}$ that breaks security of the DCS scheme. $\mathcal{B}$ can perfectly simulate the hybrids, by a similar argument to hybrid H_1 defined in Lemma B.4 – this is because the adversary only gets to see signatures on messages that satisfy C_2. Specifically, it only gets shares of $\mathcal{J} \cup \mathcal{I}$ sampled by the challenger, and $j \notin \mathcal{J} \cup \mathcal{I}$. Hence, this hybrid is indistinguishable from hybrid H_0 except with negligible probability.

Next, we define H_2 where we change the program P. Specifically, we hard-code $\mathcal{J} \cup \mathcal{I}$ and their corresponding ciphertexts, along with j in the program. Before checking any signatures, we simply set $\hat{J} \leftarrow \hat{J} \setminus \{j\}$. Moreover, for input shares of parties in $\mathcal{J} \cup \mathcal{I}$, we remove them from $\hat{J}$ if their input does not match the hard-coded ciphertext. We argue that this does not change the functionality of the program. This is because, in H_1, we constrained the public key of DCS so there is no valid signature for party j, which means that it would anyway be removed from $\hat{J}$ when the signatures are verified. Similarly, for parties in $\mathcal{J} \cup \mathcal{I}$, there is no valid signature except on the hard-coded ciphertext, so these parties would also have been removed if the input ciphertext did not match.

Let us denote $\mathsf{sh}^*_{i,j-1} = (i, (r_i, \hat{\mathsf{ct}}_{i,j-1}), \sigma^*_{i,j-1})$, for all $i \in \mathcal{I}$. In H_3, we puncture the PRF key k at all points in $T = \{r_i\}_{i\in\mathcal{I}}$, and hard-code $z_{r_i} = \mathsf{PRF.F}(k, r_i)$ in the program for all $i \in \mathcal{I}$. To decrypt any ciphertext of the form (r, ct) where $r \in T$, we now compute $\mathsf{ct} - z_r$. This does not change the functionality of the program, so H_3 is indistinguishable from H_2 by iO security.

In H_4, we sample each z_{r_i} value uniformly at random. If the r_i values are different for all shares, then, by one-time pad security, the ciphertexts $\mathsf{ct}^*_{i,j-1}$ are also random. This means we can now change from the encryptions of $(1, j-1)$ to encryptions of $(1, j)$ by simply adding $(0||1)$ to the corresponding z_{r_i} value, for all $i \in \mathcal{I}$. Observe that the functionality of the program remains the same even with this change – j is removed from $\hat{J}$ in both cases if such a ciphertext is part of the input.

This means that, behavior of the box R cannot differ by more than a negligible amount between shares $\{\mathsf{sh}^*_{i,j-1}\}$ encrypting $(1, j-1, s^*)$ and shares $\{\mathsf{sh}^*_{i,j}\}$ encrypting $(1, j, s^*)$. □ □

By Chernoff bound, we know that p'_j is within $\epsilon/16n$ of $\hat{p}_j$ for all $j \in \{0, \ldots, n\}$. Hence, Lemma B.6 implies that for all $j \in [n] \setminus (\mathcal{J} \cup \mathcal{I})$, $p'_{j-1} - p'_j \leq \epsilon/8n$ except with negligible probability. Hence, the Trace algorithm will never output any party in $[n] \setminus (\mathcal{J} \cup \mathcal{I})$.

To finish the proof, in Lemma B.7 below, we prove that the Trace will never blame any honest party in $\mathcal{I}$.

Lemma B.7. *Let $\mathcal{J}, \mathcal{I}$ and $\{\hat{p}_j\}_{j\in\{0,1,\ldots,n\}}$ be as defined above. For all $j \in \mathcal{I} \setminus \mathcal{J}$,*

$$|\hat{p}_j - \hat{p}_{j-1}| \leq \mathsf{negl}(\lambda).$$

Proof. We use a series of hybrids for the proof. Let H_0 refer to the hybrid where the challenger runs Share honestly, and then runs R with inputs $\{\mathsf{sh}^*_{i,j-1}\}_{i\in\mathcal{I}}$. Let $\mathsf{sh}^*_{j,j-1} = (j, \mathsf{ct}^*_{j,j-1}, \sigma^*_{j,j-1})$ where $\mathsf{ct}^*_{j,j-1}$ encrypts $(1, j-1)$.

We define a hybrid H_1, where the challenger samples a constrained key pair for DCS. Similar to hybrid H_1 in proof of Lemma B.5, it first samples ciphertexts for parties in $\mathcal{J} \cup \mathcal{I}$. It then runs ConstPK instead of KeyGen, with $C_1 = 1$ as defined in the Share algorithm, and C_2 which accepts input (i, ct_i) only if either (i) $i \notin \mathcal{I} \cup \mathcal{J}$, or (ii) if $i \in \mathcal{I} \cup \mathcal{J}$ then ct_i equals the ciphertexts sampled by the challenger. In particular, the only valid ciphertext for j is the one sampled by the challenger that encrypts $(1, j-1)$ (since $j \notin \mathcal{J}$). We argue that if the adversary $\mathcal{A}$ behaves differently in the two hybrids, we can use it to break security of the DCS scheme. Similar to H_1 defined in proof of Lemma B.4, we can make an adversary $\mathcal{B}$ that perfectly simulates the hybrids to $\mathcal{A}$ using its security challenger. This is because, the adversary only sees signatures on messages that satisfy C_2. Hence, the box R must behave identically in the two hybrids except with negligible probability.

Next, in hybrid H_2, we change the program P. We hard-code the sets $\mathcal{J}$ and $\mathcal{I}$ along with their ciphertexts in the program, and before checking any signatures, for any input share for $i \in \mathcal{J} \cup \mathcal{I}$, if the input ciphertext ct_i is not equal to the hard-coded ciphertexts, then we remove them from $\hat{J}$. In particular, if party $j \in J$, then it will be removed if the input ciphertext is not $\mathsf{ct}^*_{j,j-1}$. This does not change the functionality because in H_1, we constrained the public key so a valid signature exists for $\mathcal{J} \cup \mathcal{I}$ only for the hard-coded ciphertexts.

Let us denote $\mathsf{sh}^*_{i,j-1} = (i, \mathsf{ct}^*_{i,j-1} = (r_i, \hat{\mathsf{ct}}_{i,j-1}), \sigma^*_{i,j-1})$ for all $i \in \mathcal{I}$. We define another hybrid H_3, where we puncture the PRF key k at the set $T = \{r_i\}_{i\in\mathcal{I}}$. Moreover, we hard-code the PRF evaluations $z_r = \mathsf{PRF.F}(k, r)$ in the program for all $r \in T$, and we use z_r to decrypt any ciphertext of the form $(r, \cdot)$. The functionality of the program remains the same from H_2 to H_3, so the behavior of R must not change except with negligible probability.

In hybrid H_4, we sample all the z_r values uniformly at random. If no two ciphertexts in shares of $\mathcal{I}$ use the same r, then, by one-time pad security, the ciphertexts $\mathsf{ct}^*_{i,j-1}$ for all $i \in \mathcal{I}$ are distributed uniformly random as well.

In hybrid H_5, we will now change the ciphertexts to encrypt $(1, j)$ for all $i \in \mathcal{I}$, by changing z_r to $z_r + (0||1)$. We argue that doing this does not change the functionality of the program. To show that, we consider two types of inputs to the program. Note that since the ciphertexts encrypting j or $j-1$ only affect the resulting $\hat{J}$, we only need to argue that it does not change for any input, when we move from hybrid H_4 to H_5:

- $j \notin J$: The resulting $\hat{J}$ will not have j regardless of whether the ciphertexts encrypt $(1, j)$ or $(1, j-1)$, since $\hat{J} \subseteq J$ by construction.

- $j \in J$: The only valid ciphertext for j is the one encrypting $(1, j-1)$ in hybrid H_4 – since the first bit is 1, j will be in $\hat{J}$ regardless of the other ciphertexts. In this hybrid, the only valid ciphertext for j is the one encrypting $(1, j)$. Again, since the first bit is 1, regardless of the other ciphertexts, j will be in $\hat{J}$. Hence, the resulting set $\hat{J}$ is the same for the two hybrids, even for inputs containing j.

This proves that $\hat{p}_j$ and $\hat{p}_{j-1}$ cannot differ by more than a negligible amount. □ □

By a similar analysis as above, by Chernoff bound, p'_j and p'_{j-1} will also differ by a maximum of $\epsilon/8n$, and hence, honest parties in $\mathcal{I}$ will also not be blamed by the tracer. This completes the proof. □ □

Efficiency. The share size for our scheme is $\mathsf{poly}(\log(n))$. The parameter z is set to be the size of the largest circuit C_1 and C_2 defined across the security proofs – it is bounded by $O(n \log(n))$. The public parameters contain the obfuscated program, which needs to be padded to be of length $O(\mathsf{poly}(n))$ because we use programs of this size in our security proofs. Lastly, the tracing key for this scheme is of size $O(n \log(n))$ since it contains shares of all parties.

C A generic compiler for Traceable, One-time, Secret-key Encryption for general access structures

In this section, we present a generic compiler for constructing a Traceable, One-time, Secret-key Encryption for General Access structures from a traceable secret sharing scheme and a one-time secret-key encryption scheme (for simplicity, we present the compiler for one-time pad as the encryption scheme). We start with formally defining this primitive before describing our compiler.

C.1 Definition

A Traceable, One-time, Secret-key Encryption for General Access structures is a tuple of seven PPT algorithms $\mathsf{Tr\text{-}OT\text{-}SKE} := (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CombineEnc}, \mathsf{Dec}, \mathsf{CombineDec}, \mathsf{Trace}, \mathsf{Verify})$:

- $\mathsf{KeyGen}(1^\lambda, C) \to (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}, \mathsf{vk})$ is the probabilistic key-generation algorithm that takes as input the access structure represented as a monotone circuit C. It outputs public information pp, secret-keys for all n parties $\{\mathsf{sk}_i\}_{i\in[n]}$ along with a tracing key tk, and verification key vk. All other algorithms implicitly take pp as input.

- $\mathsf{Enc}(\mathsf{sk}_i, m) \to \mathsf{ct}_i$ is the probabilistic encryption algorithm which takes as input the secret key of party i and the message m and outputs a share of the encryption ct_i.

- $\mathsf{CombineEnc}(m, \{\mathsf{ct}_i\}_{i\in S}) \to \mathsf{ct}/\perp$ is the deterministic encryption combiner algorithm, which takes as input encryption shares of a set S as input and outputs a full ciphertext ct or $\perp$.

- $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \to d_i$ is the deterministic decryption algorithm. It takes in a decryptor's secret key sk_i and a ciphertext ct, and outputs a decryption share d_i.

- $\mathsf{CombineDec}(\mathsf{ct}, \{d_i\}_{i\in S}) \to m/\perp$ is the deterministic decryption combiner algorithm, which takes as input decryption shares of a set of parties S as input and outputs the full decryption m or $\perp$.

- $\mathsf{Trace}^D(\mathsf{tk}, 1^{1/\delta}, m_0, m_1, \mathsf{ct}^*, \mathcal{I}) \to (\mathcal{J}, \pi)$ is the randomized tracing algorithm. It takes as input the tracing key tk, an error parameter δ, a ciphertext ct^* and two messages m_0, m_1. It also gets oracle (black-box) access to a decoder box D along with a 'label' $\mathcal{I}$. Given these inputs, the Trace algorithm outputs a subset $\mathcal{J} \subseteq [n]$ of identities of leaking parties and an associated proof π.

- $\mathsf{Verify}(\mathsf{vk}, \mathcal{J}, \pi) \to \{0,1\}$ is the deterministic verification algorithm. It takes as input the verification key vk, an alleged traitor subset $\mathcal{J}$, and an associated proof π, and outputs either 1, implying acceptance of the proof that the parties in $\mathcal{J}$ are guilty, or 0, implying rejection.

Correctness. An honestly generated ciphertext by any authorized set of parties must be correctly decrypted by any authorized set of parties.

Definition C.1. *A Traceable, One-time, Secret-key Encryption for General Access structures scheme* Tr-OT-SKE *is said to be correct if, there exists a negligible function* $\mathsf{negl}(\lambda)$ *of the security parameter* λ*, such that, for all* $\lambda, n \in \mathbb{N}$*, all monotone circuits* C*, all subsets* $\mathcal{J}_1, \mathcal{J}_2 \subseteq [n]$ *such that* $C(x_{\mathcal{J}_1}) = C(x_{\mathcal{J}_2}) = 1$*, and all messages* m *in the message space, it holds that*

$$\Pr\left[m' = m \ : \begin{array}{c} (\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, C) \\ \mathsf{ct}_i \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{sk}_i, m)\ \forall\ i \in \mathcal{J}_1 \\ \mathsf{ct} \leftarrow \mathsf{CombineEnc}(m, \{\mathsf{ct}_i\}_{i \in \mathcal{J}_1}) \\ \forall\ i \in \mathcal{J}_2,\ d_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \\ m' \leftarrow \mathsf{CombineDec}(\mathsf{ct}, \{d_j\}_{j \in \mathcal{J}_2}) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

One-time CPA security. We require that a traceable, one-time, secret-key encryption scheme for general access structures must satisfy the notion of semantic security, as is standard for decryption for arbitrary access structures. Roughly speaking, this requires that an adversary must not be able to learn anything about the encrypted message, even if it can corrupt any unauthorized subset of parties. Note that since we are in the one-time setting, the adversary is not allowed to see encryptions or decryption shares for any ciphertext. Moreover, we consider the selective notion of security wherein the adversary chooses the set of parties it corrupts at the beginning of the game. We formalize this notion via a game based definition in Figure 26 and Definition C.2.

Games $\mathbf{ExpOTCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda)$ and $\mathbf{ExpOTTrCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda)$

1 : $(C, \mathcal{J}, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$
2 : $(\mathsf{pp}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, C)$
3 : $(m_0, m_1, \mathsf{state}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sk}_i\}_{i \in \mathcal{J}}$ **, tk, vk** $)$
4 : $b \leftarrow\!\!\$\ \{0,1\}$, choose $S \subseteq [n]$ s.t. $C(x_S) = 1$
5 : $\mathsf{ct}_i \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{sk}_i, m_b)\ \forall\ i \in S$
6 : $\mathsf{ct} \leftarrow \mathsf{CombineEnc}(m_b, \{\mathsf{ct}_i\}_{i \in S})$
7 : $b' \leftarrow\!\!\$\ \mathcal{A}(\mathsf{state}, \mathsf{ct})$
8 : **if** $C(x_{\mathcal{J}}) = 0\ \wedge b' = b$ **then return** 1 **else return** 0

Figure 26: The one-time CPA and tracer-CPA security experiments for a traceable, one-time, secret-key encryption scheme for general access structures Tr-OT-SKE and a PPT adversary $\mathcal{A}$. The difference between the two games is highlighed in blue – the adversary is also given the tracing key in the game $\mathbf{ExpTrCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda)$.

Definition C.2. *We say that* Tr-OT-SKE *satisfies one-time CPA security if for every probabilistic polynomial time adversary* $\mathcal{A}$*, the following function is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{ot-cpa}}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda) := \left| \frac{1}{2} - \Pr[\mathbf{ExpOTCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda) = 1] \right|,$$

where the game $\mathbf{ExpOTCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda)$ *is as defined in Figure 26.*

One-time Tracer CPA security. We also require a Traceable, One-time, Secret-key Encryption scheme for General Access structures to satisfy tracer semantic security, which means CPA security is maintained even against the tracer. In other words, the tracing key tk does not help in breaking secrecy. We model this in the same way as CPA security, with the only difference being that the adversary is also given the tracing key.

Definition C.3. *We say that* Tr-OT-SKE *satisfies one-time tracer CPA security if for every probabilistic polynomial time adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{ot-tr-cpa}}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda) := \left|\frac{1}{2} - \Pr[\mathbf{ExpOTTrCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda) = 1]\right|,$$

where the game $\mathbf{ExpOTTrCPA}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda)$ *is as defined in Figure 26.*

One-time Traceability. In addition, a Traceable, One-time, Secret-key Encryption scheme for General Access structures must provide one-time traceability. To formalize this notion, we generalize the $\emptyset$-strong traceability definition for TSS (presented in Section 4) to this setting. In more detail, suppose a coalition $\mathcal{J} \subseteq [n]$ of parties, such that $C(x_{\mathcal{J}}) = 0$, colludes and constructs a decoder box D using their key shares. This D is accompanied with a label $\mathcal{I} \subseteq [n]$, a ciphertext ct^* and two messages m_0, m_1. It is an algorithm that takes as input additional key shares of all parties in the set $\mathcal{I}$ and can distinguish whether the ciphertext encrypts m_0 or m_1. Intuitively, if this D is a "minimally-useful" decoder box, then it should be possible to trace it back to at least one of the parties who "contributed" its key share to it. We require that tracing D back to the corrupted parties should be done given only black-box access to it (and the label). We only consider the one-time notion wherein the adversary is not allowed to see any encryptions or decryptions before it constructs the decoder box D.

Formally, D is a minimally-useful decoder box with respect to a label $\mathcal{I}$ if it outputs the correct bit b with high probability, where ct^* is an encryption of m_b, when given as input key shares for the set $\mathcal{I}$. Definition C.4 below formally defines such reconstruction boxes. With this definition, we formalize one-time traceability in Definition C.5.

Definition C.4 (Minimally-useful Decoder Box)**.** *Let* Tr-OT-SKE *be a Traceable, One-time, Secret-key Encryption scheme for General Access structures. Let $\lambda \in \mathbb{N}$, let $C : \{0,1\}^n \to \{0,1\}$ be a monotone circuit. For $\epsilon \in [0,1]$, messages $m_0, m_1 \in \mathcal{M}_\lambda$, bit $b \leftarrow\!\!\$\ \{0,1\}$, a set $\mathcal{J} \subseteq [n]$ such that $C(x_{\mathcal{J}}) = 0$, and corresponding key shares $\{\mathsf{sk}_i\}_{i\in\mathcal{J}}$, and a label $\mathcal{I} = \{i_1, \ldots, i_f\}$, a decoder box D is $(C, \mathcal{I}, \{\mathsf{sk}_i\}_{i\in\mathcal{J}}, m_0, m_1, \mathsf{ct}^*, \epsilon)$-minimally-useful if,*

$$\Pr\left[D(\mathsf{sk}_{i_1}, \ldots, \mathsf{sk}_{i_f}) = b\right] \geq \frac{1}{2} + \epsilon,$$

where $(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, C)$, and $\mathsf{ct}_i \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{sk}_i, m_b)$ for all $i \in S$ where $S \subseteq [n]$ such that $C(x_S) = 1$, and $\mathsf{ct}^* \leftarrow \mathsf{CombineEnc}(\{\mathsf{ct}_i\}_{i\in S})$, and the probability is taken over the random coins of D.

Definition C.5. *Let $\epsilon = \epsilon(\lambda)$ be a function of the security parameter. A* Tr-OT-SKE *scheme satisfies one-time traceability if, for all* PPT *adversaries $\mathcal{A}$, the following is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{ot-trac}}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE},\epsilon}(\lambda) := \Pr\left[\mathbf{ExpOT\emptyset StrongTrace}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE},\epsilon}(\lambda) = 1\right],$$

where $\mathbf{ExpOT\emptyset StrongTrace}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE},\epsilon}(\lambda)$ *is as defined in Figure 27.*

Experiment $\mathbf{ExpOT\emptyset StrongTrace}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE},\epsilon}(\lambda)$

$(C, \mathcal{J}, m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$ // $\mathcal{A}$ chooses the parties to corrupt
$b \leftarrow\!\$ \{0,1\}$, choose $S \subseteq [n]$ s.t. $C(x_S) = 1$
$(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\$ \mathsf{KeyGen}(1^\lambda, C)$
$\mathsf{ct}_i \leftarrow\!\$ \mathsf{Enc}(\mathsf{sk}_i, m_b)\ \forall\ i \in S$
$\mathsf{ct}^* \leftarrow \mathsf{CombineEnc}(m_b, \{\mathsf{ct}_i\}_{i\in S})$
$(D, \mathcal{I}) \leftarrow\!\$ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\mathcal{J}}, \mathsf{ct}^*)$
$(\mathcal{J}^*, \pi) \leftarrow\!\$ \mathsf{Trace}^D(\mathsf{tk}, 1^{1/\epsilon}, m_0, m_1, \mathsf{ct}^*, \mathcal{I})$
$\mathsf{GoodBox} := (\mathcal{I} \cap \mathcal{J} = \emptyset) \wedge C(x_\mathcal{J}) = 0 \wedge C(x_\mathcal{I}) = 0 \wedge$
$(D \text{ is } (C, \mathcal{I}, \{\mathsf{sh}_i\}_{i\in\mathcal{J}}, m_0, m_1, \mathsf{ct}^*, \epsilon)\text{-minimally-useful})$
if $\mathcal{J}^* \not\subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 1
if $\neg\mathsf{GoodBox}$ **then return** 0
if $\mathcal{J}^* \neq \emptyset \wedge \mathcal{J}^* \subseteq \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 0 **else return** 1

Figure 27: The one-time $\emptyset$-strong tracing experiment for a Traceable, One-time, Secret-key Encryption scheme for General Access structures Tr-OT-SKE and an adversary $\mathcal{A}$.

Experiment $\mathbf{ExpNI}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}$

$(C, i^*, \mathsf{state}) \leftarrow\!\$ \mathcal{A}(1^\lambda)$
$(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{pp}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\$ \mathsf{KeyGen}(1^\lambda, C)$
$(\mathcal{J}^*, \pi) \leftarrow\!\$ \mathcal{A}(\mathsf{state}, \mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]\setminus\{i^*\}}, \mathsf{tk}, \mathsf{vk})$
if $i^* \in \mathcal{J} \wedge \mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$ **then return** 1 **else return** 0

Figure 28: The tracer non-imputability experiment for a Traceable, One-time, Secret-key Encryption scheme for General Access structures Tr-OT-SKE and an adversary $\mathcal{A}$.

Remark C.1. *Note that our notion of traceability differs significantly from related notions such as traitor tracing and threshold traitor tracing. First, we consider the simpler setting of one-time secret-key encryption (as opposed to public-key encryption). On the other hand, our definition captures arbitrary access structures beyond threshold. Last, and more importantly, we consider a scenario where an* unauthorized *subset of parties colludes to construct a decoder box. This is in contrast to (threshold) traitor tracing, where the decoder box has enough keys to decrypt.*

Tracer Non-imputability. Lastly, we also require tracer non-imputability, stating that honest parties cannot be wrongly blamed, even if the tracer is malicious. We formalize this via a security game presented in Fig. 28 and Definition C.6 below.

Definition C.6. *A* Tr-OT-SKE *scheme achieves tracer non-imputability if for all* PPT *adversaries $\mathcal{A}$, the following advantage is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{ni}}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda) := \Pr\left[\mathbf{ExpNI}_{\mathcal{A},\mathsf{Tr\text{-}OT\text{-}SKE}}(\lambda) = 1\right].$$

C.2 Our compiler

We present our compiler in Figure 29. Essentially, we sample a random secret, and generate its secret shares using the GTSS scheme. To encrypt a message, we simply use this random secret as a one-time

pad. Any authorized set of parties can encrypt by first running the reconstruction algorithm of GTSS to get the secret, and then use it as a one-time pad. Similarly, any authorized set of parties can decrypt by running the reconstruction algorithm of GTSS to recover the one-time pad. Looking ahead, we will show that all the security properties of the resulting scheme follow directly from the corresponding properties of the GTSS scheme.

Correctness. Correctness of our scheme follows trivially from correctness of the GTSS scheme. Roughly speaking, for any message m, and for any authorized set S, we know that $s' = s$ except with negligible probability (here s is the secret sampled during KeyGen and s' is as computed during Enc and CombineEnc). This means that, the resulting ciphertext of m is equal to $m \oplus s$. Similarly, for any authorized subset and any ciphertext, the value s'' computed during Dec and CombineDec is equal to s, except with negligible probability. Hence, the decryption works correctly: $\mathsf{ct} \oplus s = (m \oplus s) \oplus s = m$.

One-time CPA security. Theorem C.7 below proves that if the underlying scheme GTSS satisfies secrecy, then the resulting scheme achieves one-time CPA security.

Theorem C.7. *For any* PPT *adversary $\mathcal{A}$, there exists another* PPT *adversary $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathrm{ot-cpa}}_{\mathcal{A},\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B},\mathsf{GTSS}}(\lambda)$$

Proof. We will construct an adversary $\mathcal{B}$ that breaks secrecy for GTSS using $\mathcal{A}$. $\mathcal{A}$ starts with sending the challenge circuit C and the set of corrupt parties $\mathcal{J}$ to $\mathcal{B}$. $\mathcal{B}$ then samples $s_0, s_1 \leftarrow\!\!\$\ \mathcal{M}_\lambda$, and forwards $C, \mathcal{J}, s_0, s_1$ to its challenger.

$\mathcal{B}$ then gets back pp, along with $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$. $\mathcal{B}$ sets $\mathsf{sk}_i := \mathsf{sh}_i$ for all $i \in \mathcal{J}$, and forwards $\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\mathcal{J}}$ to $\mathcal{A}$.

$\mathcal{B}$ would then get messages m_0, m_1 from $\mathcal{A}$. It samples bits $b_1, b_2 \leftarrow\!\!\$\ \{0,1\}$, and returns $\mathsf{ct} := s_{b_1} \oplus m_{b_2}$ to $\mathcal{A}$. Let b' represent $\mathcal{A}$'s response. If $b' = b_2$, $\mathcal{B}$ responds with b_1, otherwise it responds with $\neg b_1$.

Clearly, $\mathcal{B}$ perfectly simulates the CPA game to $\mathcal{A}$ until the challenger ciphertext. We finish the proof by arguing that $\mathcal{B}$ does indeed win the secrecy game with non-negligible advantage if $\mathcal{A}$ wins its game. To see that, let us use b to denote the secret bit sampled by $\mathcal{B}$'s challenger, i.e., the shares given to $\mathcal{B}$ are shares of s_b. Let us take two cases:

- $b_1 = b$. Note that this event occurs with probability $1/2$ since $\mathcal{B}$ randomly samples b_1. If this event does occur, the ciphertext $s_{b_1} \oplus m_{b_2}$ is indeed a valid ciphertext encrypting m_{b_2}. This means that $\mathcal{A}$ must output the correct bit b_2 to win its secrecy game. Observe that if $\mathcal{A}$ outputs b_2, then $\mathcal{B}$ would respond with b_1, which is indeed the correct guess for b.
- $b_1 \neq b$. In this case, the secret s_{b_1} is distributed uniformly randomly, independent of the secret keys of parties. Hence, the ciphertext $s_{b_1} \oplus m_{b_2}$ is also distributed uniformly randomly, meaning that it contains no information about b_2, so $\mathcal{A}$ can only output b_2 with probability $\leq 1/2$, meaning that $\mathcal{B}$ outputs the wrong guess $b_1 (\neq b)$ with the same probability.

Combining the above with a standard probability analysis gives us the theorem. □

One-time Tracer CPA security. Theorem C.8 below proves that if GTSS has tracer secrecy, then the compiled scheme achieves one-time tracer CPA security.

Theorem C.8. *For any* PPT *adversary $\mathcal{A}$, there exists another* PPT *adversary $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathrm{ot-tr-cpa}}_{\mathcal{A},\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathrm{tr-sec}}_{\mathcal{B},\mathsf{GTSS}}(\lambda)$$

Proof. The proof is exactly the same as that for Theorem C.7 above. The only difference is that, $\mathcal{B}$ now also gets tk and vk from its challenger, which it simply forwards to $\mathcal{A}$. □

$\underline{\mathsf{KeyGen}(1^\lambda, C)}$:

1. Sample $s \leftarrow\!\!\$\ \mathcal{M}_\lambda$. Run $(\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in[n]}, \mathsf{tk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{GTSS.Share}(1^\lambda, s, C)$.
2. Output $\mathsf{pp}, \{\mathsf{sk}_i := \mathsf{sh}_i\}_{i\in[n]}, \mathsf{tk}, \mathsf{vk}$.

$\underline{\mathsf{Enc}(\mathsf{sk}_i, m)}$:

1. Output $\mathsf{ct}_i := \mathsf{sk}_i$.

$\underline{\mathsf{CombineEnc}(m, \{\mathsf{ct}_i\}_{i\in S})}$:

1. Parse ct_i as sk_i for all $i \in S$.
2. Run $s' \leftarrow \mathsf{GTSS.Rec}(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in S})$.
3. Output $\mathsf{ct} := m \oplus s'$.

$\underline{\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})}$:

1. Output $d_i := \mathsf{sk}_i$.

$\underline{\mathsf{CombineDec}(\mathsf{ct}, \{d_i\}_{i\in S})}$:

1. Parse d_i as sk_i for all $i \in S$.
2. Run $s'' \leftarrow \mathsf{GTSS.Rec}(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in S})$.
3. Output $\mathsf{ct} \oplus s''$.

$\underline{\mathsf{Trace}^D(\mathsf{tk}, 1^{1/\delta}, m_0, m_1, \mathsf{ct}^*, \mathcal{I})}$:

1. Compute $s_b := m_b \oplus \mathsf{ct}^*$ for $b \in \{0, 1\}$.
2. Construct a reconstruction box R as follows:
   - R takes as input secret shares of parties in $\mathcal{I}$, i.e., $\{\mathsf{sh}_i\}_{i\in\mathcal{I}}$.
   - When given such shares as input, it sets $\mathsf{sk}_i := \mathsf{sh}_i$ for all $i \in \mathcal{I}$.
   - It then outputs $D(\{\mathsf{sk}_i\}_{i\in\mathcal{I}})$.
3. Output $\mathsf{GTSS.Trace}^R(\mathsf{tk}, 1^{1/\delta}, s_0, s_1, \mathcal{I})$.

$\underline{\mathsf{Verify}(\mathsf{vk}, \mathcal{J}, \pi)}$:

1. Output $\mathsf{GTSS.Verify}(\mathsf{vk}, \mathcal{J}, \pi)$.

Figure 29: Our generic compiler to construct a Traceable, One-time, Secret-key Encryption scheme for General Access structures $\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}$ from any TSS scheme GTSS with $\emptyset$-strong traceability. For simplicity, we set the message space for the resulting scheme to be the secret space for GTSS, but this can be easily generalized to messages of arbitrary lengths using hybrid encryption.

One-time Traceability. Theorem C.9 below proves that $\emptyset$-strong traceability of GTSS directly implies one-time traceability of the compiled scheme.

Theorem C.9. *For any* PPT *adversary* $\mathcal{A}$*, there exists another* PPT *adversary* $\mathcal{B}$ *such that,*

$$\mathsf{Adv}^{\mathrm{ot-trac}}_{\mathcal{A},\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}}(\lambda) \leq \mathsf{Adv}^{\emptyset-\mathrm{str-trac}}_{\mathcal{B},\mathsf{GTSS}}(\lambda).$$

Proof. We will construct $\mathcal{B}$ that breaks $\emptyset$-strong traceability for GTSS using $\mathcal{A}$. The game starts with $\mathcal{A}$ sending the circuit C, the corrupt set $\mathcal{J}$, the challenge messages m_0, m_1.

$\mathcal{B}$ samples $\mathsf{ct}^* \leftarrow\!\!\$\ \mathcal{M}_\lambda$, and computes $s_b := \mathsf{ct}^* \oplus m_b$ for $b \in \{0,1\}$. It then forwards $C, \mathcal{J}, s_0, s_1$ to its challenger. It gets back pp, along with $\{\mathsf{sh}_i\}_{i\in\mathcal{J}}$. $\mathcal{B}$ sets $\mathsf{sk}_i := \mathsf{sh}_i$ for all $i \in \mathcal{J}$, and forwards $\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\mathcal{J}}$ along with ct^* to $\mathcal{A}$.

Eventually, $\mathcal{A}$ produces a decoder box D along with the label $\mathcal{I}$. $\mathcal{B}$ then constructs a reconstruction box R using D, exactly as in the Trace algorithm in Fig. 29.

To prove the theorem, first, we argue that if D is a minimally-useful box, then so is R. To see why, observe that the probability that R outputs the correct bit b (wherein the shares $\{\mathsf{sk}_i\}$ are secret shares of s_b) is equal to the probability that D outputs b when given shares of s_b for the set $\mathcal{I}$ as input. Now recall, $s_b = \mathsf{ct}^* \oplus m_b$, i.e., $\mathsf{ct}^* = s_b \oplus m_b$. In other words, if the secret keys are shares of s_b, then ct^* encrypts m_b. Hence, if D is minimally-useful, then it must output b with probability $1/2 + \epsilon$. This means that R also outputs the correct bit b with the same probability.

To finish the proof, we now observe that if $\mathcal{A}$ wins its one-time traceability game, then so does $\mathcal{B}$. To see why, observe that $\mathcal{B}$ simply forwards the box R as constructed in the Trace algorithm to its challenger. Also recall that the tracer in our compiler simply runs $\mathsf{GTSS.Trace}$ on this box R. Hence, if the tracer for the $\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}$ outputs $\mathcal{J}^*, \pi$, the challenger for $\mathcal{B}$ will also compute the same $\mathcal{J}^*, \pi$. Moreover, the Verify of $\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}$ is also identical to that of GTSS. This means that, if $\mathcal{A}$ wins, i.e., $\mathcal{J}^* = \emptyset$, or $\mathcal{J}^* \not\subseteq \mathcal{J}$ or $\mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 0$, then $\mathcal{B}$ also wins. □

Tracer Non-imputability. Tracer Non-imputability of GTSS trivially implies tracer non-imputability of the compiled scheme. Theorem C.10 formalizes this.

Theorem C.10. *For all* PPT *adversaries* $\mathcal{A}$*, there exists another* PPT *adversary* $\mathcal{B}$ *such that,*

$$\mathsf{Adv}^{\mathrm{ni}}_{\mathcal{A},\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{ni}}_{\mathcal{B},\mathsf{GTSS}}(\lambda).$$

Proof. $\mathcal{B}$ acts as the challenger to $\mathcal{A}$. The game starts with $\mathcal{A}$ sending the circuit C and the party i^* to $\mathcal{B}$. $\mathcal{B}$ then samples a random secret $s \leftarrow\!\!\$\ \mathcal{M}_\lambda$, and sends C, i^*, s to its challenger.

$\mathcal{B}$ then gets back $\mathsf{pp}, \{\mathsf{sh}_i\}_{i\in[n]\setminus\{i^*\}}$ along with tk, vk from its challenger. $\mathcal{B}$ then sets $\mathsf{sk}_i := \mathsf{sh}_i$ for all $i \in [n] \setminus \{i^*\}$, and forwards $\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]\setminus\{i^*\}}, \mathsf{tk}, \mathsf{vk}$ to $\mathcal{A}$.

$\mathcal{B}$ eventually receives $\mathcal{J}^*, \pi$ from $\mathcal{A}$, and it simply forwards these to its challenger.

Clearly, $\mathcal{B}$ perfectly simulates the tracer non-imputability game to $\mathcal{A}$. Moreover, if $\mathcal{A}$ wins, then $i^* \in \mathcal{J}^*$, and $\Pi_{\mathsf{Tr\text{-}OT\text{-}SKE}}.\mathsf{Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = \mathsf{GTSS.Verify}(\mathsf{vk}, \mathcal{J}^*, \pi) = 1$, which is exactly the winning condition for $\mathcal{B}$. Hence, $\mathcal{B}$ has the same advantage as $\mathcal{A}$. □

D Constructing Doubly Constrained Signature

We now present a construction of doubly-constrained signatures based on indistinguishability obfuscation and puncturable PRFs.

At a high level, we sample a PRF key in KeyGen, and the signature on a message m is simply the PRF evaluation at m. We use an obfuscated program for verification, which has the PRF key hard-coded, and checks if the signature matches the PRF evaluation at m. To constrain the signing key for a circuit C_1, we simply use an obfuscated program with the PRF key hard-coded, which outputs the PRF evaluation only

$\underline{\mathsf{KeyGen}(1^\lambda, \ell, z, C_1)}$:

1. Sample PRF key $k \leftarrow\!\!\$\ \{0,1\}^\lambda$. Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a pseudo-random generator.
2. Output $\mathsf{sk} \leftarrow \mathsf{PRF.F}(k, \cdot)$, $\mathsf{pk} \leftarrow \mathsf{Obf}(\mathsf{P}_{ver})$ and $\mathsf{sk}_{C_1} \leftarrow \mathsf{Obf}(\mathsf{P}_{const,C_1})$. / Both sk and sk_{C_1} are programs that take messages as input and output a signature.

$\underline{\mathsf{P}_{ver}(m, \sigma)}$:

1. **Constants:** k
2. Output 1 if $\mathsf{PRG}(\mathsf{PRF.F}(k, m)) = \mathsf{PRG}(\sigma)$. Otherwise, output 0.

$\underline{\mathsf{P}_{const,C}(m)}$:

1. **Constants:** k, C.
2. If $C(m) = 0$, output $\perp$.
3. Otherwise, output $\mathsf{PRF.F}(k, m)$.

$\underline{\mathsf{Sign}(\mathsf{sk}, m)}$:

1. Parse sk as P, and output $\mathsf{P}(m)$.

$\underline{\mathsf{SignVerify}(\mathsf{pk}, m, \sigma_m)}$:

1. Parse pk as $\hat{\mathsf{P}}$ and output $\hat{\mathsf{P}}(m, \sigma_m)$.

$\underline{\mathsf{ConstPK}(1^\lambda, \ell, z, C_1, C_2)}$:

1. Sample PRF key $k \leftarrow\!\!\$\ \{0,1\}^\lambda$. Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a pseudo-random generator.
2. Let $C_{12} := C_1$ if $C_1 \implies C_2$, otherwise, $C_{12} \leftarrow C_1 \wedge C_2$.
3. Output $\mathsf{sk} \leftarrow \mathsf{PRF.F}(k, \cdot)$, $\mathsf{pk} \leftarrow \mathsf{Obf}(\mathsf{P}_{ver,C_2})$ and $\mathsf{sk}_{C_1} \leftarrow \mathsf{Obf}(\mathsf{P}_{const,C_{12}})$.

$\underline{\mathsf{P}_{ver,C}(m, \sigma)}$:

1. **Constants:** k, C.
2. Output 0 if $C(m) = 0$.
3. Otherwise, output 1 if $\mathsf{PRG}(\mathsf{PRF.F}(k, m)) = \mathsf{PRG}(\sigma)$. Otherwise, output 0.

Figure 30: Our doubly-constrained signature scheme iODCS built from indistinguishability obfuscation (iO) and puncturable PRFs.

if $C_1(m) = 1$. Lastly, to constrain the public key for a circuit C_2, we modify the verification obfuscated program, to simply abort if the circuit C_2 is not satisfied at m. Figure 30 presents the full scheme.

Correctness. Correctness of our construction follows directly from correctness of obfuscation. For keys generated via KeyGen, for all messages, the signature would be $\mathsf{PRF.F}(k, m)$, hence the program $\mathsf{Obf}(\mathsf{P}_{ver})$ will indeed output 1. Similarly, for all messages that satisfy C_1, the obfuscation of P_{const,C_1} will indeed

output the correct PRF evaluation, and hence this signature would also pass verification. By a similar argument, correctness also holds for keys generated via ConstPK.

Second, for any message m such that $C_2(m) = 0$, the program $\mathsf{Obf}(\mathsf{P}_{ver})$ will never output 1 by correctness of obfuscation. Hence, there is no valid signature for such messages.

Identical-distribution of Signatures. This property trivially follows from correctness of the obfuscated program P_{const,C_1}. Specifically, for any message m such that $C_1(m) = 1$, the signature generated by both sk and the constrained signing key sk_{C_1} is simply $\mathsf{PRF.F}(k, m)$. Theorem D.1 formally proves this.

Theorem D.1. *For all adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathrm{id-dist}}_{\mathcal{A},\mathsf{iODCS}}(\lambda) = 0,$$

if Obf *has perfect correctness.*

Proof. For any message m such that $C_1(m) = 1$, the signature generated from sk is $\mathsf{PRF.F}(k, m)$. On the other hand, when the program $\mathsf{Obf}(\mathsf{P}_{const,C_1})$ is run with such a message as input, it will also output the same value i.e., $\mathsf{PRF.F}(k, m)$, by correctness of obfuscation. The theorem follows. □

Security. Theorem D.2 below proves security of our iO-based DCS construction. We use complexity leveraging, i.e., we rely on sub-exponential security of obfuscation, PRF and the PRG.

Theorem D.2. *For all* PPT *adversaries $\mathcal{A}$, there exist* PPT *adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_4$, $\mathcal{B}_5$, $\mathcal{B}_6$, $\mathcal{B}_7$, $\mathcal{B}_8$, $\mathcal{B}_9$ such that,*

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathcal{A},\mathsf{iODCS}}(\lambda) \leq 2^{\ell}\cdot\Big(\begin{array}{c}\frac{1}{2^{\lambda-1}} + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_2,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_3,\mathsf{PRF}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_4,\mathsf{PRG}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_5,\mathsf{Obf}}(\lambda) \\ + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_6,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_7,\mathsf{PRF}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_8,\mathsf{PRG}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_9,\mathsf{Obf}}(\lambda)\end{array}\Big)$$

Proof. Let E_1 denote the event that the circuit C_1 sent by the adversary is more restrictive than C_2, i.e., $C_1 \implies C_2$.

By total probability,

$$\Pr[\mathbf{G}^{sec}_{\mathcal{A},\mathsf{WIDCS}} = 1] = \Pr[(\mathbf{G}^{sec}_{\mathcal{A},\mathsf{WIDCS}} = 1) \wedge \mathsf{E}_1] + \Pr[(\mathbf{G}^{sec}_{\mathcal{A},\mathsf{WIDCS}} = 1) \wedge \neg\mathsf{E}_1]$$

We now bound each of these terms separately. Let us start with the case where $C_1 \implies C_2$. We define a series of hybrids:

H_0: This is the original security game for DCS, wherein the challenger uses the unconstrained key, i.e., runs KeyGen with C_1.

For each $i \in \{0, \ldots, 2^{\ell} - 1\}$, we define the following hybrids:

$H_{i,0}$: When the challenger gets circuits C_1 and C_2, it constructs another circuit $C_{2,i}$ where, $C_{2,i}(x) = C_2(x)$ for all $x \leq i$ (treating x as an integer in $\{0, \ldots, 2^{\ell} - 1\}$), and $C_{2,i}(x) = 1$ for all $x > i$. It then runs ConstPK with input C_1 and $C_{2,i}$, and gives the resulting $\mathsf{pk}_{C_{2,i}}$ to the adversary. Note that, in $H_{0,0}$, the program $C_{2,0}$ essentially outputs 1 for all programs. Moreover, the constrained key sk_{C_1} is identical in hybrids H_0 and $H_{0,0}$ – it contains an obfuscated program with C_1 and the PRF key k hard-coded. Hence, this hybrid is equivalent to H_0.

$H_{i,1}$: If $C_2(i+1) = 1$, the challenger behaves exactly as in $H_{i,0}$. Otherwise, the challenger changes the program P_{const,C_1} to contain a punctured PRF key. Specifically, it computes $k_{i+1} = \mathsf{PRF.Punc}(k, \{i+1\})$, and hard-codes k_{i+1} (instead of k) in P_{const,C_1}. We argue that this does not change functionality of this program. To see this, observe that $C_2(i + 1) = 0$ implies that $C_1(i + 1) = 0$. Hence, the program P_{const,C_1} would never need to evaluate the PRF at the value $i + 1$. Hence, if the adversary can distinguish between hybrids $H_{i,0}$ and $H_{i,1}$, it can be used to construct another adversary $\mathcal{B}_{1,i}$ that breaks security of obfuscation.

$H_{i,2}$: If $C_2(i+1) = 1$, the challenger behaves exactly as in $H_{i,1}$. Otherwise, the challenger uses the constrained key k_{i+1} also in the verification program. It also hard-codes $y = \mathsf{PRG}(\mathsf{PRF.F}(k, i+1))$ in the program. Figure 31 depicts the full program. The challenger sends as pk the obfuscation of $\mathsf{P}^{(2)}_{ver,C_i}$. Note that this program has the same functionality as the verification program in $H_{i,1}$. Hence, if the adversary can distinguish between $H_{i,1}$, $H_{i,2}$, we can use it to make an adversary $\mathcal{B}_{2,i}$ that breaks security of obfuscation.

$H_{i,3}$: If $C_2(i+1) = 1$, the challenger behaves exactly as in $H_{i,2}$. Otherwise, it sets $y = \mathsf{PRG}(s)$ for a uniformly random seed $s \leftarrow\!\!\$\ \{0,1\}^\lambda$ (and everything else is the same as $H_{i,2}$). Note that, in this game, the view of the adversary only contains the punctured PRF key k_{i+1}. Moreover, this is enough to simulate the full game for the challenger because the adversary can anyway never query the signature on $i+1$ as $C_2(i+1) = 0$. Therefore, if $\mathcal{A}$ can distinguish hybrid $H_{i,3}$ from hybrid $H_{i,2}$, then we can use it to construct an adversary $\mathcal{B}_{3,i}$ for the punctured PRF with the same advantage as $\mathcal{A}$.

$H_{i,4}$: If $C_2(i+1) = 1$, the challenger behaves exactly as in $H_{i,3}$. Otherwise, it behaves exactly as in $H_{i,3}$, except that it samples $y \leftarrow\!\!\$\ \{0,1\}^{2\lambda}$ uniformly from the range of PRG. If $\mathcal{A}$ can detect this change, then we can construct an adversary $\mathcal{B}_{4,i}$ for the PRG.

Now, observe that y is in the image of the PRG only with probability $1/2^\lambda$. Hence, if $C_2(i+1) = 0$ and y is not in the image of the PRG, then the verification program will always output 0 for input $i+1$ – i.e., it has the same functionality as the verification program in hybrid $H_{i+1,0}$, which also outputs 0 for $i+1$. This is true even if $C_2(i+1) = 1$, in which case, both the programs just compare the input to $\mathsf{PRG}(\mathsf{PRF.F}(k, i+1))$. Hence, for both $C_2(i+1) = 0$ and $C_2(i+1) = 1$, the obfuscated verification programs in hybrids $H_{i,4}$ and $H_{i+1,0}$ compute the same functionality.

Moreover, the functionality of the constrained signing program is also the same for the two hybrids. To see that, consider two cases: (i) input $m \neq i+1$, in this case, the signing program in both hybrids does the same thing: check if C_1 is satisfied, if so, output the PRF evaluation. By correctness of the puncturing, the evaluation gotten from the punctured key is equal to the real evaluation, for any $m \neq i+1$. (ii) $m = i+1$: Observe that because $C_2(i+1) = 0$, this implies $C_1(i+1) = 0$, hence, the constrained signing programs in both hybrids will simply output $\perp$ on this input. If $C_2(i+1) = 1$, then the programs are actually identical in both hybrids.

Hence, if $\mathcal{A}$ can distinguish between hybrids $H_{i,4}$ and $H_{i+1,0}$ with probability ϵ, then we can use it to construct $\mathcal{B}_{5,i}$ to break security of Obf with advantage $\epsilon - 1/2^\lambda$.

Finally, notice that the hybrid $H_{2^\ell-1,0}$ is identical to the case where the challenger runs $\mathsf{ConstPK}$ and gives the constrained public key to the adversary. Combining $\mathcal{B}_{1,i}$, $\mathcal{B}_{2,i}$, $\mathcal{B}_{3,i}$, $\mathcal{B}_{4,i}$, $\mathcal{B}_{5,i}$ into single adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_4$, $\mathcal{B}_5$, which guess i at random, we get,

$$\Pr[(\mathbf{G}^{sec}_{\mathcal{A},\mathsf{WIDCS}} = 1) \wedge \mathsf{E}_1] \leq 2^\ell \cdot (2^{-\lambda} + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_1,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_2,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_3,\mathsf{PRF}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_4,\mathsf{PRG}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_5,\mathsf{Obf}}(\lambda))$$

$\mathsf{P}^{(2)}_{ver,C_i}(m, \sigma)$:

1. **Constants:** k_{i+1}, C_i, i, y.
2. Output 0 if $C_i(m) = 0$.
3. If $m = i+1$, then output 1 if $\mathsf{PRG}(\sigma) = y$.
4. If $m \neq i+1$, output 1 if $\mathsf{PRG}(\mathsf{PRF.F}(k, m)) = \mathsf{PRG}(\sigma)$. Otherwise, output 0.

Figure 31: The verification program used in hybrid $H_{i,2}$ in the security proof of iODCS.

Next, we consider the event $\neg \mathsf{E}_1$, wherein $C_1 \not\Rightarrow C_2$. This case is simpler because the adversary does not get the constrained signing key as input. We again consider a series of hybrids.

$\hat{H}_0$: This is the original security game, wherein the challenger samples the unconstrained signing key.

Similar to above, we define a series of hybrids, for each $i \in \{0, \ldots, 2^\ell - 1\}$:

$\hat{H}_{i,0}$: Similar to $H_{i,0}$ above, the challenger constructs circuit $C_{2,i}$ which is $= C_2(x)$ for all $x \leq i$ and 1 for all other inputs. It then runs $\mathsf{ConstPK}$ with input C_1 and $C_{2,i}$ and gives the resulting keys to $\mathcal{A}$. Note that, $\hat{H}_{0,0}$ is equivalent to hybrid $\hat{H}_0$.

$\hat{H}_{i,1}$: Similar to $H_{i,2}$ above, if $C_2(i+1) = 1$, the challenger behaves exactly as in $\hat{H}_{i,0}$. Otherwise, the challenger samples a constrained key k_{i+1}, and also hard-codes $y = \mathsf{PRG}(\mathsf{PRF.F}(k, i+1))$ in the verification program. Figure 31 depicts the full program. As argued in $H_{i,2}$ above, if $\mathcal{A}$ can distinguish between $\hat{H}_{i,0}$ and $\hat{H}_{i,1}$, we can use it to make $\mathcal{B}_{6,i}$ that breaks security of obfuscation.

$\hat{H}_{i,2}$: Similar to hybrid $H_{i,3}$ above, if $C_2(i+1) = 1$, the challenger behaves exactly as in $\hat{H}_{i,1}$. Otherwise, it sets $y = \mathsf{PRG}(s)$ for a uniformly random seed $s \leftarrow\!\!\$\ \{0,1\}^\lambda$ (and everything else is the same as $H_{i,2}$). As argued in hybrid $H_{i,3}$ above, if $\mathcal{A}$ can distinguish between $\hat{H}_{i,1}$ and $\hat{H}_{i,2}$, we can use it to make $\mathcal{B}_{7,i}$ that breaks security of the punctured PRF.

$\hat{H}_{i,3}$: Similar to hybrid $H_{i,4}$ above, if $C_2(i+1) = 1$, the challenger behaves exactly as in $\hat{H}_{i,1}$. Otherwise, it behaves as in $\hat{H}_{i,2}$, but samples $y \leftarrow\!\!\$\ \{0,1\}^{2\lambda}$. If $\mathcal{A}$ can detect this change, we can use it to construct an adversary $\mathcal{B}_{8,i}$ that breaks PRG security.

Now, observe that y is in the image of the PRG only with probability $1/2^\lambda$. Following the same argument as in hybrid $H_{i,4}$ above, the obfuscated verification program in $\hat{H}_{i,3}$ is functionally equivalent to that in hybrid $\hat{H}_{i+1,0}$. Hence, if $\mathcal{A}$ can distinguish between hybrids $\hat{H}_{i,3}$ and $\hat{H}_{i+1,0}$ with probability ϵ, then we can use it to construct $\mathcal{B}_{9,i}$ to break security of Obf with advantage $\epsilon - 1/2^\lambda$.

Finally, notice that the hybrid $\hat{H}_{2^\ell-1,0}$ is identical to the case where the challenger runs $\mathsf{ConstPK}$ with circuits C_1, C_2. Combining $\mathcal{B}_{6,i}$,$\mathcal{B}_{7,i}$,$\mathcal{B}_{8,i}$,$\mathcal{B}_{9,i}$, into single adversaries $\mathcal{B}_6$, $\mathcal{B}_6$, $\mathcal{B}_6$, $\mathcal{B}_6$, which guess i at random, we get,

$$\Pr[(\mathbf{G}^{sec}_{\mathcal{A},\mathsf{WIDCS}} = 1) \wedge \neg \mathsf{E}_1] \leq 2^\ell \cdot (2^{-\lambda} + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_6,\mathsf{Obf}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_7,\mathsf{PRF}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_8,\mathsf{PRG}}(\lambda) + \mathsf{Adv}^{\mathrm{sec}}_{\mathcal{B}_9,\mathsf{Obf}}(\lambda)).$$

This proves the theorem. □