A Note on Non-interactive Deniable Secret Sharing from Indistinguishability Obfuscation

Shafik Nassar
UT Austin
shafik@cs.utexas.edu

November 12, 2026

Abstract

Deniable secret sharing, introduced by Canetti et al. (TCC 2025), protects shareholders when a subset of them is coerced into revealing their shares. Namely, a set of fakers can produce fake shares that are consistent with a fake secret of their choice, and that are indistinguishable from real shares. Very recently, Amores et al. (TCC 2026) studied deniable secret sharing in a setting where the fakers cannot interact to produce their fake shares. Specifically, each faker locally computes its fake share given only its own share, the set of fakers, and the fake secret. Their constructions of non-interactive deniable secret sharing either have shares of exponential size or rely on virtual black-box obfuscation.

We show how to upgrade any secret sharing scheme, for any access structure, into a non-interactive deniable secret sharing scheme using a collusion-resistant constrained PRF with static security, which can be instantiated from polynomially-secure indistinguishability obfuscation and one-way functions. Our transformation incurs an additive share size overhead of $\mathrm{poly}(\lambda, n)$, where λ is the security parameter and n is the number of shareholders. By using indistinguishability obfuscation directly, together with a somewhere-statistically binding hash, we further reduce the overhead to $\mathrm{poly}(\lambda, \log n)$. In combination with prior work on succinct computational secret sharing, we get succinct deniable secret sharing from indistinguishability obfuscation and standard assumptions.

1 Introduction

Secret sharing allows a dealer to split a secret message into n shares, and give each share to a distinct shareholder. Correctness requires that a qualified subset of shareholders can combine their shares to recover the secret, while privacy requires that any unqualified subset learns nothing about the secret. In a recent work, Canetti et al. [CDK+25] defined the notion of *deniable secret sharing*, which requires meaningful security even in the presence of an adversary that can coerce *honest* users to reveal their shares. Specifically, the coerced users coordinate in order to produce *fake* shares on some other fake secret. The first basic security we property require is that fake shares are indistinguishable from real shares. This property is called "Fake-Real Indistinguishability". The second property is the classic privacy property of secret sharing, which should hold even if the unqualified shares are complemented by fake shares from all of the other users.

Very recently, Amores et al. [ADC+26] considered the notion of *non-interactive* faking, where the fakers do not exchange any messages. The only coordination is that they know who they are, and what message they want to fake on. That is, there exists an efficient faking algorithm that inputs a set of fakers F, a single share for user $i \in F$ and a fake message, and outputs a fake share for user i on that fake message. This notion seems difficult because the fakers should agree on a lie without really coordinating, yet other users cannot compute that same lie, otherwise the adversary can distinguish fake shares from real shares. In the same work, Amores et al. give a construction from PRGs with an exponential share size. The idea there is to give every user a prepared fake share (or enough information to derive it) for every possible faker set that it could be a part of. In addition, they propose an approach using obfuscation, which they can only instantiate using virtual black-box obfuscation [BGI+01]. The idea is to have a single

obfuscated program that helps the fakers coordinate their fake shares, and essentially compresses the exponential information into an obfuscated program. However, virtual black-box obfuscation for general programs does not exist, and the authors explicitly leave a construction from *indistinguishability obfuscation* ($i\mathcal{O}$) [BGI+01, GGH+13] as an open problem.

1.1 Our Results

In this note, we rely on collusion-resistant constrained pseudorandom functions [BW13, KPTZ13, BGI14] to upgrade any classic secret sharing scheme into a (non-interactive) deniable secret sharing scheme, thus resolving the open problem posed by Amores et al. [ADC+26]. That is, we prove the following theorem:

Theorem 1.1. *Assume the existence of a polynomially-secure collusion-resistant constrained pseudorandom function and a secret sharing scheme for access structure Γ. Then there exists a non-interactive deniable secret sharing scheme for the same access structure Γ.*

Notably, we only require a collusion-resistant constrained pseudorandom function with *static security*, which can be instantiated from *polynomially-secure* $i\mathcal{O}$ and one-way functions [BZ14]. We remark that the transformation from Theorem 1.1 incurs $\mathrm{poly}(\lambda, n)$ additive overhead over in the share size, where λ is the security parameter and n is the number of shareholders (users) in the system.

We push things further and present a transformation with $\mathrm{poly}(\lambda, \log n)$ additive overhead using standard $i\mathcal{O}$ techniques and a somewhere-statistically binding hash [HW15]. We sketch the approach in Section 4, and provide an AI-generated proof in Appendix A. We take full responsibility for the correctness of the proofs in Appendix A. That is, the following theorem is also proven:

Theorem 1.2. *Assume the existence of polynomially-secure indistinguishability obfuscation and a somewhere-statistically binding hash. Additionally, assume the existence of a secret sharing scheme for access structure Γ. Then there exists a non-interactive deniable secret sharing scheme for the same access structure Γ with an additive share size overhead of $\mathrm{poly}(\lambda, \log n)$, where λ is the security parameter and n is the number of users.*

This is especially useful for upgrading succinct computational secret sharing [ABI+23] while preserving succinctness. Specifically, as a corollary of our work and the previous work on succinct computational secret sharing from $i\mathcal{O}$ [LNW25, JJMP25], we get succinct deniable secret sharing from $i\mathcal{O}$ and standard assumptions.

1.2 Technical Overview

The conceptual idea underlying the prior work [ADC+26] is to give the fakers access to an oracle that helps them coordinate their lie. The problem is that if everyone has access to that oracle, then the adversary can derive that same lie and distinguish a real share from a fake share. As such, we want to limit the access to the oracle: it only outputs the fake shares for a set F to users in that set.

The approach we take is as follows: each user i gets their own oracle K_i, which on input a set of fakers F outputs the coordinated lie for that set. However, this oracle K_i only outputs the information if $i \in F$, otherwise it aborts. By inspecting this briefly, we observe that this is achievable via a constrained pseudorandom function [BW13, KPTZ13, BGI14]. Recall that a constrained PRF has the same syntax as a normal PRF, but it is also equipped with a constraining algorithm that takes a master key and a predicate $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, and outputs a constrained key $K^{(C)}$ that evaluates the original PRF only on inputs $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ that satisfy $C(x) = 1$. Collusion-resistant security stipulates that given any number of constrained keys $K^{(C_1)}, \ldots, K^{(C_t)}$, the evaluation of the master key on any x such that $C_i(x) = 0$ for all $i \in [t]$ is pseudorandom.

To coordinate the lie, the fakers need some randomness to derive consistent shares. Crucially, the randomness needs to be the same for all fakers in the set. As such, when generating the shares, the dealer also generates a (master) PRF key K, where the function maps a set F to a string r that is long enough to be used for the sharing algorithm, and now each set of fakers F uses $r = \mathsf{PRF}(K, F)$ to coordinate their fake shares. The dealer now gives each user i a constrained PRF key K_i that can only be used to evaluate $\mathsf{PRF}(K, F)$ if $i \in F$. One additional point is that since the real shares include *consistent* constrained PRF keys, the fake shares should as well. However, we cannot supply the

same constrained keys since they can be used to check if the shares are fake. Instead, we modify the output length of the constrained PRF to be long enough to also derive a new master PRF key, which each faker can independently constrain. The security reduces directly to the collusion-resistant security of the constrained PRF. Our construction and the formal proof are in Section 3.

Acknowledgement of AI Use

The ideas in this note were generated by the authors. AI was used for copy editing of the main body. In addition, the construction and proofs in Appendix A were completely generated by an AI agent that was given as input the construction in Section 3 and the overview in Section 4. The authors verified the content of this paper and take full responsibility for its correctness.

2 Preliminaries

For any positive integer $n \in \mathbb{N}$, we denote $[n] \coloneqq \{1, \ldots, n\}$. The security parameter will be denoted by λ. For a binary string $x \in \{0,1\}^*$, we denote the length of x by $|x|$. A function $\varepsilon(\lambda)$ is called negligible if for every positive constant c, we have $\varepsilon(\lambda) = o(\lambda^{-c})$. A function $f(\lambda)$ is polynomial if there exists a positive constant c such that $f(\lambda) = O(\lambda^c)$. We denote this by writing $f(\lambda) = \mathsf{poly}(\lambda)$. For any set S, when we write $x \xleftarrow{R} S$ we mean that x is sampled uniformly from S. We represent subsets of $[n]$ via n-bit strings in the natural way, where $i \in S$ if and only if the i^{th} bit in the corresponding string is 1.

2.1 Secret Sharing

Definition 2.1 (Access Structure)**.** Fix $n \in \mathbb{N}$. A collection of subsets $\Gamma \subseteq 2^{[n]}$ is *monotone* if for all $A \in \Gamma$ and all $B \subseteq [n]$ such that $A \subseteq B$, we have $B \in \Gamma$. An *access structure for* $[n]$ is a non-empty monotone collection Γ.

Definition 2.2 (Computational Secret Sharing)**.** Let Γ be an access structure for $[n]$. A computational secret sharing (CSS) scheme for Γ and message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{CSS}} = (\mathsf{Share}, \mathsf{Rec})$ with the following syntax:

- $\mathsf{Share}(1^\lambda, m) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$: On input a security parameter λ and a secret $m \in \mathcal{M}$, the share algorithm outputs shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$.
- $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S}) \to m$: On input a set of parties $S \subseteq [n]$ and their corresponding shares $\{\mathsf{sh}_i\}_{i \in S}$, the reconstruction algorithm outputs a secret $m \in \mathcal{M} \cup \{\bot\}$, where $\bot$ indicates a reconstruction error.

A computational secret sharing scheme should satisfy the following properties:

- **Correctness:** For any $\lambda \in \mathbb{N}$, any secret $m \in \mathcal{M}$, and any authorized set $S \in \Gamma$, we have

$$\Pr\left[\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S}) = m : (\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m)\right] = 1.$$

- **Privacy:** For any adversary $\mathcal{A}$, define the secret sharing security game with the parameter b as follows:
  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses an unauthorized set $S \subseteq [n]$ with $S \notin \Gamma$ and a pair of secrets $m_0, m_1 \in \mathcal{M}$, and sends (S, m_0, m_1) to the challenger.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and sends $\{\mathsf{sh}_i\}_{i \in S}$ to $\mathcal{A}$.
  3. The adversary outputs a guess $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{CSS} satisfies privacy if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

A *deniable* secret sharing scheme [CDK+25] extends a secret sharing scheme with a faking algorithm. We consider *non-interactive* faking: each shareholder in a set $F \subseteq [n]$, called the *fakers*, locally converts its own real share into a fake share, knowing only the set F and a target fake secret m^{fake}, such that the fake shares of F are consistent with m^{fake}. The remaining shareholders $S = [n] \setminus F$, called the *snitches*, reveal their real shares. For security, we focus on fake-real indistinguishability, in addition to privacy.

Definition 2.3 (Deniable Secret Sharing [CDK+25])**.** Let Γ be an access structure for $[n]$. A deniable secret sharing (DSS) scheme for Γ and message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\text{DSS}} = (\mathsf{Share}, \mathsf{Rec}, \mathsf{Fake})$, where Share and Rec have the same syntax as in Definition 2.2, and the non-interactive faking algorithm Fake has the following syntax:

- $\mathsf{Fake}(F, i, \mathsf{sh}_i, m^{\text{fake}}) \to \widetilde{\mathsf{sh}}_i$: On input a set of fakers $F \subseteq [n]$, an index $i \in F$ of a faker, its share sh_i, and a fake secret $m^{\text{fake}} \in \mathcal{M}$, the non-interactive faking algorithm outputs a fake share $\widetilde{\mathsf{sh}}_i$. Each faker $i \in F$ runs Fake locally and independently, without interacting with the other fakers.

A deniable secret sharing scheme should satisfy the following properties:

- **Correctness:** (Share, Rec) satisfies correctness as in Definition 2.2.
- **Privacy:** For any adversary $\mathcal{A}$, define the privacy game with the parameter b as follows:
  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses a set of fakers $F \subseteq [n]$ such that the set of snitches $S = [n] \setminus F$ is unauthorized (i.e., $S \notin \Gamma$), and secrets $m_0, m_1, m^{\text{fake}} \in \mathcal{M}$, and sends $(F, m_0, m_1, m^{\text{fake}})$ to the challenger.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and fake shares $\widetilde{\mathsf{sh}}_i \leftarrow \mathsf{Fake}(F, i, \mathsf{sh}_i, m^{\text{fake}})$ for each $i \in F$, and sends $\left\{\widetilde{\mathsf{sh}}_i\right\}_{i \in F}$ and $\{\mathsf{sh}_i\}_{i \in S}$ to $\mathcal{A}$.
  3. The adversary outputs a guess $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that Π_{DSS} satisfies privacy if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that
  $$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$
- **Fake-real indistinguishability:** For any adversary $\mathcal{A}$, define the fake-real indistinguishability game with the parameter b as follows:
  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses a partition (P_0, P_1) of $[n]$ and secrets $m_0, m_1 \in \mathcal{M}$, and sends (P_0, P_1, m_0, m_1) to the challenger.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and fake shares $\widetilde{\mathsf{sh}}_i \leftarrow \mathsf{Fake}(P_{1-b}, i, \mathsf{sh}_i, m_{1-b})$ for each $i \in P_{1-b}$. It sets $\widetilde{\mathsf{sh}}_i = \mathsf{sh}_i$ for each $i \in P_b$, and sends $\left\{\widetilde{\mathsf{sh}}_i\right\}_{i \in [n]}$ to $\mathcal{A}$.
  3. The adversary outputs a guess $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that Π_{DSS} satisfies fake-real indistinguishability if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that
  $$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

In the fake-real indistinguishability game, the shares of P_0 are always consistent with m_0 and the shares of P_1 are always consistent with m_1. The bit b only determines which of the two sets holds real shares and which set faked.

2.2 Building Block

Our basic construction requires only a *collusion-resistant constrained pseudorandom function* [BW13, KPTZ13, BGI14]. In our application, it suffices to rely on *static security*, therefore we can instantiate this primitive from polynomially-secure $i\mathcal{O}$ and polynomially secure one-way functions [BZ14].

Definition 2.4 (Constrained Pseudorandom Function [BW13, KPTZ13, BGI14])**.** A constrained pseudorandom function for a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{CPRF}} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to K$: On input a security parameter λ, an input length ℓ_{in}, and an output length ℓ_{out}, the key generation algorithm outputs an evaluation key K.
- $\mathsf{Eval}(K, x) \to y$: On input an evaluation key K (which may be a master key or a constrained key) and a string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\bot\}$.
- $\mathsf{Constrain}(K, C) \to K^{(C)}$: On input a key K, and a binary circuit $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ in the class $\mathcal{C}_\lambda$, the constraining algorithm outputs a constrained evaluation key $K^{(C)}$.

A constrained pseudorandom function should satisfy the following properties:

- **Constrained correctness:** For any $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any Boolean circuit $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ in $\mathcal{C}_\lambda$, any K in the support of $\mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, and any $K^{(C)}$ in the support of $\mathsf{Constrain}(K, C)$, we have

$$\mathsf{Eval}(K^{(C)}, x) = \begin{cases} \mathsf{Eval}(K, x) & C(x) = 1 \\ \bot & C(x) = 0 \end{cases}$$

- **Static collusion-resistant security:** For any adversary $\mathcal{A}$, define the constrained security game with the parameter b as follows:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses an input length $1^{\ell_{\mathsf{in}}}$, an output length $1^{\ell_{\mathsf{out}}}$, an input $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, and t binary circuits $C_1, \ldots, C_t : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, and sends $(1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, x^*, C_1, \ldots, C_t)$ to the challenger.
  2. If there exists any $i \in [t]$ such that $C_i(x^*) = 1$, the experiment aborts and outputs a random bit.
  3. The challenger samples a key $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ and a constrained key $K^{(C_i)} \leftarrow \mathsf{Constrain}(K, C_i)$ for each $i \in [t]$, and sends $K^{(C_1)}, \ldots, K^{(C_t)}$ to $\mathcal{A}$.
  4. The challenger additionally computes the challenge $y_0 = \mathsf{Eval}(K, x^*)$, samples $y_1 \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$ and sends y_b to $\mathcal{A}$.
  5. The adversary outputs a guess b', which is the output of the experiment.

  We say that Π_{CPRF} satisfies constrained pseudorandomness if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

3 Supporting Deniability Using Constrained PRFs

Construction 3.1. Our construction uses only a secret sharing scheme $\Pi_{\mathsf{CSS}} = (\mathsf{Share}, \mathsf{Rec})$ and a constrained PRF $\Pi_{\mathsf{CPRF}} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain})$. We assume that $\mathsf{Share}(1^\lambda, \cdot)$ and $\mathsf{KeyGen}(1^\lambda, \cdot, \cdot)$ always use λ random bits each, independent of the other parameters. This is possible by expanding the λ-bit string using a PRG.

We will now describe our algorithms $(\mathsf{Share}', \mathsf{Rec}', \mathsf{Fake})$.

- $\mathsf{Share}'(1^\lambda, m)$:

1. Samples shares using the underlying secret sharing $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m)$.
2. Samples a master PRF key $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda})$.
3. For each $i \in [n]$, computes a constrained key $K_i \leftarrow \mathsf{Constrain}(K, C_i)$, where $C_i(x_1, \ldots, x_n) \coloneqq x_i$ gets a set description x and accepts only if i is in the set.
4. Sets $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$ for each $i \in [n]$.
5. Outputs $\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n$.

- $\mathsf{Rec}'(S, \{\mathsf{sh}'_i\}_{i \in S})$:
  1. Parses $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$ for each $i \in S$.
  2. Outputs $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S})$, ignoring the constrained keys.
- $\mathsf{Fake}(F, i, \mathsf{sh}'_i, m^{\mathsf{fake}})$:
  1. Parses $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$.
  2. Lets $x_F \in \{0,1\}^n$ be the description of the set F, namely, the j-th bit of x_F is 1 if and only if $j \in F$.
  3. Derives pseudorandomness $\rho \| r = \mathsf{Eval}(K_i, x_F)$, where $\rho, r \in \{0,1\}^\lambda$. Since $i \in F$, we have $C_i(x_F) = 1$, so K_i evaluates x_F to the same value as the master key K, which is common to all fakers in F.
  4. Samples a new master PRF key $\widetilde{K} = \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda}; \rho)$ using the randomness ρ.
  5. Samples new shares $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m^{\mathsf{fake}}; r)$ using the randomness r.
  6. Computes a constrained key $\widetilde{K}_i \leftarrow \mathsf{Constrain}(\widetilde{K}, C_i)$.
  7. Outputs $\widetilde{\mathsf{sh}}'_i = (\widetilde{\mathsf{sh}}_i, \widetilde{K}_i)$.

Theorem 3.2. *Assume that Π_{CSS} satisfies correctness (as per Definition 2.2). Then Construction 3.1 satisfies correctness (as per Definition 2.3).*

Proof. Fix $\lambda \in \mathbb{N}$, a secret $m \in \mathcal{M}$, and an authorized set $S \in \Gamma$. Let $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n) \leftarrow \mathsf{Share}'(1^\lambda, m)$. By construction, for each $i \in [n]$ we have $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$, where $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ is sampled from $\mathsf{Share}(1^\lambda, m)$. The algorithm $\mathsf{Rec}'(S, \{\mathsf{sh}'_i\}_{i \in S})$ ignores the constrained keys $\{K_i\}_{i \in S}$ and outputs $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S})$. Since $S \in \Gamma$, by the correctness of Π_{CSS} we have $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S}) = m$ with probability 1. Therefore, $\mathsf{Rec}'(S, \{\mathsf{sh}'_i\}_{i \in S}) = m$ with probability 1. □

Remark 3.3 (Share Size)**.** Note that the new share sh'_i consists of an original share sh_i and a constrained PRF key K_i on a bit-fixing circuit. By the efficiency of the constrained PRF, the size of the key is $|K_i| = \mathsf{poly}(\lambda, n)$, therefore this construction has an additive overhead for the share size of $\mathsf{poly}(\lambda, n)$. This is enough to establish a feasibility result, since the overhead is only polynomial. In Section 4, we sketch a construction that achieves $\mathsf{poly}(\lambda, \log n)$ additive overhead using stronger tools.

3.1 Security Analysis

Throughout this section, for a set $F \subseteq [n]$ we denote by $x_F \in \{0,1\}^n$ its description, namely, the j-th bit of x_F is 1 if and only if $j \in F$. Recall that $C_i(x_F) = 1$ if and only if $i \in F$.

Theorem 3.4 (Privacy)**.** *Assume that Π_{CSS} is a secure computational secret sharing scheme (as per Definition 2.2), and that Π_{CPRF} is a constrained PRF for a circuit class that contains $C_1, \ldots, C_n$ and satisfies static collusion-resistant security (as per Definition 2.4). Then Construction 3.1 satisfies privacy (as per Definition 2.3).*

Let $\mathcal{A}$ be an adversary for the privacy game. We define the following sequence of hybrids starting with the real privacy game Real^b.

- Real^b: This is the real privacy game with parameter b.

1. On input 1^λ, the adversary $\mathcal{A}$ sends a set of fakers $F \subseteq [n]$ such that $S = [n] \setminus F$ is unauthorized, and secrets $m_0, m_1, m^{\mathsf{fake}} \in \mathcal{M}$.
2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and a master key $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda})$.
3. For each $i \in S$, the challenger samples a constrained key $K_i \leftarrow \mathsf{Constrain}(K, C_i)$ and sets $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$.
4. The challenger computes $\rho \| r = \mathsf{Eval}(K, x_F)$, where $\rho, r \in \{0,1\}^\lambda$.
5. The challenger computes $\widetilde{K} = \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda}; \rho)$ and $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m^{\mathsf{fake}}; r)$.
6. For each $i \in F$, the challenger samples $\widetilde{K}_i \leftarrow \mathsf{Constrain}(\widetilde{K}, C_i)$ and sets $\widetilde{\mathsf{sh}}'_i = (\widetilde{\mathsf{sh}}_i, \widetilde{K}_i)$.
7. The challenger sends $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in F}$ and $\left\{\mathsf{sh}'_i\right\}_{i \in S}$ to $\mathcal{A}$.
8. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

By the constrained correctness of Π_{CPRF}, for every $i \in F$ we have $\mathsf{Eval}(K_i, x_F) = \mathsf{Eval}(K, x_F)$, since $C_i(x_F) = 1$. Hence, all fakers derive the same pseudorandomness $\rho \| r = \mathsf{Eval}(K, x_F)$, and this is exactly the privacy game of Definition 2.3 instantiated with Construction 3.1. Note that the challenger does not need the constrained keys K_i of the fakers $i \in F$, since $\mathcal{A}$ only receives their fake shares.

- Hyb^b: same as Real^b, except that the pseudorandomness of the fakers is replaced with true randomness.
  4. The challenger samples $\rho \| r \xleftarrow{\mathrm{R}} \{0,1\}^{2\lambda}$.

Lemma 3.5. *Assume that Π_{CPRF} satisfies static collusion-resistant security. Then for any efficient adversary $\mathcal{A}$ and any $b \in \{0,1\}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. We construct an adversary $\mathcal{B}$ for the static collusion-resistant security game of Π_{CPRF} as follows:

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ and gets $(F, m_0, m_1, m^{\mathsf{fake}})$. Let $S = [n] \setminus F$.
2. Algorithm $\mathcal{B}$ sends $(1^n, 1^{2\lambda}, x_F, \{C_i\}_{i \in S})$ to the Π_{CPRF} challenger.
3. The Π_{CPRF} challenger responds with constrained keys $\{K_i\}_{i \in S}$ and a challenge y.
4. Algorithm $\mathcal{B}$ samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and sets $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$ for each $i \in S$.
5. Algorithm $\mathcal{B}$ parses $\rho \| r = y$ and computes $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in F}$ exactly as in steps 5 and 6 of Real^b.
6. Algorithm $\mathcal{B}$ sends $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in F}$ and $\left\{\mathsf{sh}'_i\right\}_{i \in S}$ to $\mathcal{A}$, and outputs whatever bit b' that $\mathcal{A}$ outputs.

For each $i \in S$ we have $i \notin F$, hence $C_i(x_F) = 0$ and the Π_{CPRF} experiment never aborts. If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. If $y = \mathsf{Eval}(K, x_F)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Real}^b(\mathcal{A})$, and if y is uniformly random, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b(\mathcal{A})$. Therefore, the advantage of $\mathcal{B}$ is exactly

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]\right|,$$

and the lemma follows by the static collusion-resistant security of Π_{CPRF}. □

Lemma 3.6. *Assume that Π_{CSS} is secure. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Since KeyGen and Share use exactly λ random bits each, and $\rho \| r$ is uniform in Hyb^b, the master key $\widetilde{K} = \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda}; \rho)$ and the shares $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m^{\mathsf{fake}}; r)$ are distributed as fresh samples of $\mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda})$ and $\mathsf{Share}(1^\lambda, m^{\mathsf{fake}})$, independently of b, of $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ and of K. Therefore, the only part of the view of $\mathcal{A}$ that depends on b is $\{\mathsf{sh}_i\}_{i \in S}$. We construct an adversary $\mathcal{B}$ for the security game of Π_{CSS} as follows:

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ and gets $(F, m_0, m_1, m^{\mathsf{fake}})$. Let $S = [n] \setminus F$.
2. Algorithm $\mathcal{B}$ sends (S, m_0, m_1) to the Π_{CSS} challenger and gets back $\{\mathsf{sh}_i\}_{i \in S}$.
3. Algorithm $\mathcal{B}$ samples $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda})$ and $K_i \leftarrow \mathsf{Constrain}(K, C_i)$ for each $i \in S$, and sets $\mathsf{sh}'_i = (\mathsf{sh}_i, K_i)$.
4. Algorithm $\mathcal{B}$ samples $\rho \| r \xleftarrow{\mathrm{R}} \{0,1\}^{2\lambda}$ and computes $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in F}$ exactly as in Hyb^b.
5. Algorithm $\mathcal{B}$ sends $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in F}$ and $\left\{\mathsf{sh}'_i\right\}_{i \in S}$ to $\mathcal{A}$, and outputs whatever bit b' that $\mathcal{A}$ outputs.

Since $S \notin \Gamma$, algorithm $\mathcal{B}$ is a valid adversary for the security game of Π_{CSS}. If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. If the Π_{CSS} challenger shares m_b, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b(\mathcal{A})$. Therefore, the advantage of $\mathcal{B}$ is exactly

$$\left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right|,$$

and the lemma follows by the security of Π_{CSS}. □

Proof of Theorem 3.4. Let $\mathcal{A}$ be an efficient adversary for the privacy game against Construction 3.1. By Lemma 3.5 and Lemma 3.6 and the triangle inequality, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

This concludes the proof of Theorem 3.4.

Theorem 3.7 (Fake-Real Indistinguishability)**.** *Assume that Π_{CPRF} is a constrained PRF for a circuit class that contains $C_1, \ldots, C_n$ and satisfies static collusion-resistant security (as per Definition 2.4). Then Construction 3.1 satisfies fake-real indistinguishability (as per Definition 2.3).*

Let $\mathcal{A}$ be an adversary for the fake-real indistinguishability game. We define the following sequence of hybrids starting with the real fake-real indistinguishability game Real^b.

- Real^b: This is the real fake-real indistinguishability game with parameter b.
  1. On input 1^λ, the adversary $\mathcal{A}$ sends a partition (P_0, P_1) of $[n]$ and secrets $m_0, m_1 \in \mathcal{M}$.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and a master key $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda})$.
  3. For each $i \in P_b$, the challenger samples a constrained key $K_i \leftarrow \mathsf{Constrain}(K, C_i)$ and sets $\widetilde{\mathsf{sh}}'_i = (\mathsf{sh}_i, K_i)$.
  4. The challenger computes $\rho \| r = \mathsf{Eval}(K, x_{P_{1-b}})$, where $\rho, r \in \{0,1\}^\lambda$.
  5. The challenger computes $\widetilde{K} = \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda}; \rho)$ and $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m_{1-b}; r)$.
  6. For each $i \in P_{1-b}$, the challenger samples $\widetilde{K}_i \leftarrow \mathsf{Constrain}(\widetilde{K}, C_i)$ and sets $\widetilde{\mathsf{sh}}'_i = (\widetilde{\mathsf{sh}}_i, \widetilde{K}_i)$.
  7. The challenger sends $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in [n]}$ to $\mathcal{A}$.
  8. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  By the constrained correctness of Π_{CPRF}, for every $i \in P_{1-b}$ we have $\mathsf{Eval}(K_i, x_{P_{1-b}}) = \mathsf{Eval}(K, x_{P_{1-b}})$, since $C_i(x_{P_{1-b}}) = 1$. Hence, all fakers derive the same pseudorandomness $\rho \| r = \mathsf{Eval}(K, x_{P_{1-b}})$, and this is exactly the fake-real indistinguishability game of Definition 2.3 instantiated with Construction 3.1.

- Hyb^b: same as Real^b, except that the pseudorandomness of the fakers is replaced with true randomness.

    4. The challenger samples $\rho \| r \xleftarrow{R} \{0,1\}^{2\lambda}$.

Lemma 3.8. *Assume that Π_{CPRF} satisfies static collusion-resistant security. Then for any efficient adversary $\mathcal{A}$ and any $b \in \{0,1\}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. We construct an adversary $\mathcal{B}$ for the static collusion-resistant security game of Π_{CPRF} as follows:

1. Algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ and gets (P_0, P_1, m_0, m_1).
2. Algorithm $\mathcal{B}$ sends $(1^n, 1^{2\lambda}, x_{P_{1-b}}, \{C_i\}_{i \in P_b})$ to the Π_{CPRF} challenger.
3. The Π_{CPRF} challenger responds with constrained keys $\{K_i\}_{i \in P_b}$ and a challenge y.
4. Algorithm $\mathcal{B}$ samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$ and sets $\widetilde{\mathsf{sh}}_i' = (\mathsf{sh}_i, K_i)$ for each $i \in P_b$.
5. Algorithm $\mathcal{B}$ parses $\rho \| r = y$ and computes $\left\{\widetilde{\mathsf{sh}}_i'\right\}_{i \in P_{1-b}}$ exactly as in steps 5 and 6 of Real^b.
6. Algorithm $\mathcal{B}$ sends $\left\{\widetilde{\mathsf{sh}}_i'\right\}_{i \in [n]}$ to $\mathcal{A}$, and outputs whatever bit b' that $\mathcal{A}$ outputs.

Since (P_0, P_1) is a partition, for each $i \in P_b$ we have $i \notin P_{1-b}$, hence $C_i(x_{P_{1-b}}) = 0$ and the Π_{CPRF} experiment never aborts. If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. If $y = \mathsf{Eval}(K, x_{P_{1-b}})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Real}^b(\mathcal{A})$, and if y is uniformly random, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b(\mathcal{A})$. Therefore, the advantage of $\mathcal{B}$ is exactly

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]\right|,$$

and the lemma follows by the static collusion-resistant security of Π_{CPRF}. □

Lemma 3.9. *For any (unbounded) adversary $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1].$$

Proof. Since KeyGen and Share use exactly λ random bits each, and $\rho \| r$ is uniform in Hyb^b, the master key $\widetilde{K} = \mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda}; \rho)$ and the shares $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m_{1-b}; r)$ are distributed as fresh samples of $\mathsf{KeyGen}(1^\lambda, 1^n, 1^{2\lambda})$ and $\mathsf{Share}(1^\lambda, m_{1-b})$, independently of $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ and of K. Therefore, the view of $\mathcal{A}$ in Hyb^b is distributed as follows:

- For each $i \in P_b$, the share $\widetilde{\mathsf{sh}}_i'$ consists of the i-th share of a fresh sharing of m_b and a constrained key $\mathsf{Constrain}(K, C_i)$ for a fresh master key K.
- For each $i \in P_{1-b}$, the share $\widetilde{\mathsf{sh}}_i'$ consists of the i-th share of a fresh sharing of m_{1-b} and a constrained key $\mathsf{Constrain}(\widetilde{K}, C_i)$ for a fresh master key $\widetilde{K}$.

Moreover, the two sharings and the two master keys are mutually independent. In other words, for every $\beta \in \{0,1\}$, the shares of P_β consist of a fresh sharing of m_β together with constrained keys of a fresh master key, independently of the shares of $P_{1-\beta}$. This distribution does not depend on b, and the lemma follows. □

Proof of Theorem 3.7. Let $\mathcal{A}$ be an efficient adversary for the fake-real indistinguishability game against Construction 3.1. By Lemma 3.8 and Lemma 3.9 and the triangle inequality, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

This concludes the proof of Theorem 3.7.

4 Deniability with Small Overhead

As noted in Remark 3.3, the overhead in Construction 3.1 is polynomial in the number of users. We show how to get rid of that dependence by explicitly using $i\mathcal{O}$ and a somewhere-statistically binding hash [HW15].

First, it is helpful to view the constrained key K_i in the previous construction as an obfuscated program that gets a set S, and only agrees to evaluate a PRF on S if $i \in S$. Recall that the set is represented by an n-bit string. Our goal is to compress this obfuscated program by compressing the input. The natural approach is to hash the set S and evaluate the PRF on the digest. Importantly, the program must refuse to evaluate if $i \notin S$. To enforce this, we require the evaluator to additionally provide a local opening of position i to the value 1, thus proving that the set represented by the digest indeed contains the element i.

For both security properties of privacy and fake-real indistinguishability, we rely on puncturable PRFs. Namely, both security arguments pass if we manage to puncture all the PRF keys in the snitches' shares on the "target digest" dig^*, where dig^* is the hash of the faker set F. This can be done on a program-by-program basis using a somewhere-statistically binding hash. Namely, to puncture the PRF key in program K_i on dig^*, we first make the hash statistically binding on index i. Recall that i is a snitch, so by definition it is not in F. By the somewhere-statistical binding property, the digest dig^* cannot be opened to 1 in position i since there is already an opening to 0 in that position (just because dig^* is a hash of a string that has 0 in that position). This means that we can replace the PRF key with a punctured key without changing the functionality of the program, therefore the change goes unnoticed by the adversary via $i\mathcal{O}$ security. Once all the PRF keys in the programs are punctured on dig^*, we can appeal to the punctured security to argue that the fakers derived a pseudorandom value that suffices for security.

Since the hash key has size $\mathrm{poly}(\lambda, \log n)$, and so does every digest and opening, we conclude that the size of the obfuscated program is also $\mathrm{poly}(\lambda, \log n)$. As a result, we get the desired $\mathrm{poly}(\lambda, \log n)$ additive overhead. The full proof was generated by AI and is presented in Appendix A.

References

[ABI+23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1553–1566. ACM, 2023.

[ADC+26] Ignacio Amores, Tamar Ben David, Matteo Campanelli, Sebastian Kolby, Eran Omri, Anat Paskin-Cherniavsky, and Sophia Yakoubov. Deniable secret sharing: Towards practical faking. In *TCC*, 2026.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *International workshop on public key cryptography*, pages 501–519. Springer, 2014.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II 19*, pages 280–300. Springer, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CDK+25] Ran Canetti, Ivan Damgård, Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Deniable secret sharing. In *TCC*, 2025.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[JJMP25] Abhishek Jain, Zhengzhong Jin, Surya Mathialagan, and Omer Paneth. On succinct obfuscation via propositional proofs. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1703–1740. IEEE, 2025.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684, 2013.

[LNW25] George Lu, Shafik Nassar, and Brent Waters. Succinct computational secret sharing for monotone circuits. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 99–129. Springer, 2025.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT*, 2015.

A Deniability with Small Overhead: Full Construction and Proofs

In this section, we formally present and prove the construction sketched in Section 4. We first define the building blocks in Appendix A.1. We then present the construction in Appendix A.2 and prove its security in Appendix A.3.

A.1 Building Blocks

Definition A.1 (Indistinguishability Obfuscation [BGI+12, GGH+13])**.** Let $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of polynomial-size circuits. An indistinguishability obfuscator $i\mathcal{O}$ for $\mathcal{C}$ is an efficient algorithm that takes as input a security parameter λ and a circuit $C \in C_\lambda$, and outputs a circuit C'. An indistinguishability obfuscator should satisfy the following properties:

- **Functionality preserving:** For all $\lambda \in \mathbb{N}$, all $C \in C_\lambda$, and all inputs x, we have

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

- **Security:** For any adversary $\mathcal{A}$, define the security game with the parameter b as follows:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses two circuits $C_0, C_1 \in C_\lambda$ of the same size such that $C_0(x) = C_1(x)$ for all inputs x, and sends (C_0, C_1) to the challenger.
  2. The challenger samples $C' \leftarrow i\mathcal{O}(1^\lambda, C_b)$ and sends C' to $\mathcal{A}$.
  3. The adversary outputs a guess $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that $i\mathcal{O}$ satisfies security if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition A.2 (Puncturable PRF [BW13, KPTZ13, BGI14])**.** A puncturable pseudorandom function is a tuple of efficient algorithms $\Pi_{\mathsf{PPRF}} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Puncture})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to K$: On input a security parameter λ, an input length ℓ_{in}, and an output length ℓ_{out}, the key generation algorithm outputs a key K.

- $\mathsf{Puncture}(K, x^*) \to K^{(x^*)}$: On input a key K and a point $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, the puncture algorithm outputs a punctured key $K^{(x^*)}$.
- $\mathsf{Eval}(K, x) \to y$: On input a key K (which may be a master key or a punctured key) and a string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\perp\}$. This algorithm is deterministic.

A puncturable pseudorandom function should satisfy the following properties:

- **Functionality preserving:** For any $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, any K in the support of $\mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, any $K^{(x^*)}$ in the support of $\mathsf{Puncture}(K, x^*)$, and any $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $x \neq x^*$, we have

$$\mathsf{Eval}(K^{(x^*)}, x) = \mathsf{Eval}(K, x).$$

- **Punctured pseudorandomness:** For any adversary $\mathcal{A}$, define the punctured pseudorandomness game with the parameter b as follows:

  1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses an input length $1^{\ell_{\mathsf{in}}}$, an output length $1^{\ell_{\mathsf{out}}}$, and a point $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, and sends $(1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, x^*)$ to the challenger.
  2. The challenger samples a key $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ and a punctured key $K^{(x^*)} \leftarrow \mathsf{Puncture}(K, x^*)$, and sends $K^{(x^*)}$ to $\mathcal{A}$.
  3. The challenger computes $y_0 = \mathsf{Eval}(K, x^*)$, samples $y_1 \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$, and sends y_b to $\mathcal{A}$.
  4. The adversary outputs a guess $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{PPRF} satisfies punctured pseudorandomness if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition A.3 (Somewhere Statistically Binding Hash [HW15])**.** A somewhere statistically binding (SSB) hash for binary strings is a tuple of efficient algorithms $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, n, i^*) \to \mathsf{hk}$: On input a security parameter λ, an input length n (in binary), and a binding index $i^* \in [n]$, the setup algorithm outputs a hash key hk.
- $\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{dig}$: On input a hash key hk and a string $x \in \{0,1\}^n$, the hash algorithm outputs a digest dig. This algorithm is deterministic.
- $\mathsf{Open}(\mathsf{hk}, x, i) \to \pi$: On input a hash key hk, a string $x \in \{0,1\}^n$, and an index $i \in [n]$, the opening algorithm outputs an opening π. This algorithm is deterministic.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, u, \pi) \to \beta$: On input a hash key hk, a digest dig, an index $i \in [n]$, a bit $u \in \{0,1\}$, and an opening π, the verification algorithm outputs a bit $\beta \in \{0,1\}$. This algorithm is deterministic.

An SSB hash should satisfy the following properties:

- **Efficiency:** For all $\lambda, n \in \mathbb{N}$, all $i^* \in [n]$, all hk in the support of $\mathsf{Setup}(1^\lambda, n, i^*)$, all $x \in \{0,1\}^n$, and all $i \in [n]$, the hash key hk, the digest $\mathsf{Hash}(\mathsf{hk}, x)$, and the opening $\mathsf{Open}(\mathsf{hk}, x, i)$ have size at most $\mathsf{poly}(\lambda, \log n)$, and Verify runs in time $\mathsf{poly}(\lambda, \log n)$. We denote by $\ell_{\mathsf{dig}} = \ell_{\mathsf{dig}}(\lambda, n)$ the length of the digest.
- **Correctness:** For all $\lambda, n \in \mathbb{N}$, all $i^* \in [n]$, all $x = x_1 \cdots x_n \in \{0,1\}^n$, and all $i \in [n]$, we have

$$\Pr\left[\mathsf{Verify}\big(\mathsf{hk}, \mathsf{Hash}(\mathsf{hk}, x), i, x_i, \mathsf{Open}(\mathsf{hk}, x, i)\big) = 1 : \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n, i^*)\right] = 1.$$

- **Index hiding:** For any adversary $\mathcal{A}$, define the index hiding game with the parameter b as follows:

1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses an input length n and two indices $i_0, i_1 \in [n]$, and sends (n, i_0, i_1) to the challenger.
2. The challenger samples $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n, i_b)$ and sends hk to $\mathcal{A}$.
3. The adversary outputs a guess $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{SSB} satisfies index hiding if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Somewhere statistical binding:** There exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda, n \in \mathbb{N}$ and all $i^* \in [n]$,

$$\Pr\left[\begin{array}{c} \exists\, \mathsf{dig}, \pi_0, \pi_1 \text{ such that} \\ \mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i^*, 0, \pi_0) = 1 \ \wedge \ \mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i^*, 1, \pi_1) = 1 \end{array} : \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n, i^*)\right] \leq \mathsf{negl}(\lambda).$$

We say that a hash key hk is *binding on* i^* if there do not exist $\mathsf{dig}, \pi_0, \pi_1$ as above.

Puncturable PRFs exist assuming one-way functions [BW13, KPTZ13, BGI14], and SSB hash exists under various standard assumptions [HW15, OPWW15].

A.2 The Construction

Throughout this section, for a set $F \subseteq [n]$ we denote by $x_F \in \{0,1\}^n$ its description, namely, the j-th bit of x_F is 1 if and only if $j \in F$.

Construction A.4. Our construction uses the following building blocks:

- A secret sharing $\Pi_{\mathsf{CSS}} = (\mathsf{Share}, \mathsf{Rec})$ for Γ and $\mathcal{M}$.
- An SSB hash $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ with digest length $\ell_{\mathsf{dig}} = \ell_{\mathsf{dig}}(\lambda, n)$.
- A puncturable PRF $\Pi_{\mathsf{PPRF}} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Puncture})$.
- An indistinguishability obfuscator $i\mathcal{O}$.

We assume that $\mathsf{Share}(1^\lambda, \cdot)$, $\mathsf{SSB.Setup}(1^\lambda, n, \cdot)$, and $\mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$ always use λ random bits each, independent of the other parameters. This is possible by expanding the λ-bit string using a PRG. Our construction uses the program $\mathsf{Prog}[K, \mathsf{hk}, i]$, described in Figure 1. When obfuscating Prog, we pad it to the maximum size of the programs that appear in the security proof (i.e., where K is replaced with a punctured key).

Constants: A PPRF key K, an SSB hash key hk, and an index $i \in [n]$.
Inputs: A digest dig and an opening π.

1. If $\mathsf{SSB.Verify}(\mathsf{hk}, \mathsf{dig}, i, 1, \pi) = 0$, output $\perp$.
2. Otherwise, output $\mathsf{PPRF.Eval}(K, \mathsf{dig})$.

Figure 1: Program $\mathsf{Prog}[K, \mathsf{hk}, i]$.

We will now describe our algorithms $(\mathsf{Share}', \mathsf{Rec}', \mathsf{Fake})$.

- $\mathsf{Share}'(1^\lambda, m)$:
  1. Samples shares using the underlying secret sharing $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m)$.

2. Samples an SSB hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, 1)$.
3. Samples a master PPRF key $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$.
4. For each $i \in [n]$, samples an obfuscated program $P_i \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Prog}[K, \mathsf{hk}, i])$, where Prog is described in Figure 1.
5. Sets $\mathsf{sh}'_i = (\mathsf{sh}_i, \mathsf{hk}, P_i)$ for each $i \in [n]$.
6. Outputs $\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n$.

- $\mathsf{Rec}'(S, \{\mathsf{sh}'_i\}_{i \in S})$:
  1. Parses $\mathsf{sh}'_i = (\mathsf{sh}_i, \mathsf{hk}_i, P_i)$ for each $i \in S$.
  2. Outputs $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S})$, ignoring the hash keys and the obfuscated programs.
- $\mathsf{Fake}(F, i, \mathsf{sh}'_i, m^{\mathsf{fake}})$:
  1. Parses $\mathsf{sh}'_i = (\mathsf{sh}_i, \mathsf{hk}, P_i)$.
  2. Computes the digest $\mathsf{dig} = \mathsf{SSB.Hash}(\mathsf{hk}, x_F)$ and the opening $\pi_i = \mathsf{SSB.Open}(\mathsf{hk}, x_F, i)$.
  3. Derives pseudorandomness $y = P_i(\mathsf{dig}, \pi_i)$ and parses it as $y = \rho_{\mathsf{hk}} \| \rho_K \| r$, where $\rho_{\mathsf{hk}}, \rho_K, r \in \{0,1\}^\lambda$.
  4. Samples a new SSB hash key $\widetilde{\mathsf{hk}} = \mathsf{SSB.Setup}(1^\lambda, n, 1; \rho_{\mathsf{hk}})$ using the randomness ρ_{hk}.
  5. Samples a new master PPRF key $\widetilde{K} = \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda}; \rho_K)$ using the randomness ρ_K.
  6. Samples new shares $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m^{\mathsf{fake}}; r)$ using the randomness r.
  7. Samples an obfuscated program $\widetilde{P}_i \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Prog}[\widetilde{K}, \widetilde{\mathsf{hk}}, i])$.
  8. Outputs $\widetilde{\mathsf{sh}}'_i = (\widetilde{\mathsf{sh}}_i, \widetilde{\mathsf{hk}}, \widetilde{P}_i)$.

We first observe that all fakers in F derive the same pseudorandomness, and hence the same $\widetilde{\mathsf{hk}}$, $\widetilde{K}$, and $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n)$. Indeed, all fakers hold the same hash key hk, and since SSB.Hash is deterministic, they all compute the same digest dig. For each $i \in F$, the i-th bit of x_F is 1, so by the correctness of Π_{SSB} we have $\mathsf{SSB.Verify}(\mathsf{hk}, \mathsf{dig}, i, 1, \pi_i) = 1$. Therefore, by the functionality preserving of $i\mathcal{O}$, we get $P_i(\mathsf{dig}, \pi_i) = \mathsf{PPRF.Eval}(K, \mathsf{dig})$, which is the same for all $i \in F$.

Theorem A.5. *Assume that Π_{CSS} satisfies correctness (as per Definition 2.2). Then Construction A.4 satisfies correctness (as per Definition 2.3).*

Proof. Fix $\lambda \in \mathbb{N}$, a secret $m \in \mathcal{M}$, and an authorized set $S \in \Gamma$. Let $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n) \leftarrow \mathsf{Share}'(1^\lambda, m)$. By construction, for each $i \in [n]$ we have $\mathsf{sh}'_i = (\mathsf{sh}_i, \mathsf{hk}, P_i)$, where $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ is sampled from $\mathsf{Share}(1^\lambda, m)$. The algorithm $\mathsf{Rec}'(S, \{\mathsf{sh}'_i\}_{i \in S})$ ignores the hash keys and the obfuscated programs, and outputs $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S})$. Since $S \in \Gamma$, by the correctness of Π_{CSS} we have $\mathsf{Rec}(S, \{\mathsf{sh}_i\}_{i \in S}) = m$ with probability 1. Therefore, $\mathsf{Rec}'(S, \{\mathsf{sh}'_i\}_{i \in S}) = m$ with probability 1. □

Remark A.6 (Share Size)**.** The new share sh'_i consists of an original share sh_i, a hash key hk, and an obfuscated program P_i. By the efficiency of Π_{SSB}, we have $|\mathsf{hk}| = \mathsf{poly}(\lambda, \log n)$ and $\ell_{\mathsf{dig}} = \mathsf{poly}(\lambda, \log n)$. The program $\mathsf{Prog}[K, \mathsf{hk}, i]$ (even when padded) has constants of size $\mathsf{poly}(\lambda, \log n)$, runs SSB.Verify in time $\mathsf{poly}(\lambda, \log n)$, and evaluates a PPRF with input length ℓ_{dig} and output length 3λ in time $\mathsf{poly}(\lambda, \ell_{\mathsf{dig}}) = \mathsf{poly}(\lambda, \log n)$. Note that the output length of the PPRF does not depend on the amount of randomness that Share would otherwise consume, since we assume that it uses λ random bits. Since $i\mathcal{O}$ incurs a polynomial blowup, we get $|P_i| = \mathsf{poly}(\lambda, \log n)$. Therefore, Construction A.4 has an additive overhead for the share size of $\mathsf{poly}(\lambda, \log n)$.

A.3 Security Analysis

Theorem A.7 (Privacy). *Assume that* Π_{CSS} *is a secure computational secret sharing scheme (as per Definition 2.2),* Π_{SSB} *satisfies index hiding and somewhere statistical binding (as per Definition A.3),* Π_{PPRF} *satisfies punctured pseudorandomness (as per Definition A.2), and* $i\mathcal{O}$ *is a secure indistinguishability obfuscator (as per Definition A.1). Then Construction A.4 satisfies privacy (as per Definition 2.3).*

Let $\mathcal{A}$ be an adversary for the privacy game. Throughout the proof, we denote by $S = \{s_1, \ldots, s_t\}$ the set of snitches chosen by $\mathcal{A}$, where $s_1 < \cdots < s_t$, and we set $s_0 := 1$. We define the following sequence of hybrids starting with the real privacy game Real^b.

- Real^b: This is the real privacy game with parameter b.

  1. On input 1^λ, the adversary $\mathcal{A}$ sends a set of fakers $F \subseteq [n]$ such that $S = [n] \setminus F$ is unauthorized, and secrets $m_0, m_1, m^{\mathsf{fake}} \in \mathcal{M}$.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$, a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, 1)$, and a master key $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$.
  3. The challenger computes the target digest $\mathsf{dig}^* = \mathsf{SSB.Hash}(\mathsf{hk}, x_F)$.
  4. For each $i \in S$, the challenger samples $P_i \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Prog}[K, \mathsf{hk}, i])$ and sets $\mathsf{sh}'_i = (\mathsf{sh}_i, \mathsf{hk}, P_i)$.
  5. The challenger computes $y = \mathsf{PPRF.Eval}(K, \mathsf{dig}^*)$ and parses $y = \rho_{\mathsf{hk}} \| \rho_K \| r$.
  6. The challenger computes $\widetilde{\mathsf{hk}} = \mathsf{SSB.Setup}(1^\lambda, n, 1; \rho_{\mathsf{hk}})$, $\widetilde{K} = \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda}; \rho_K)$ and $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m^{\mathsf{fake}}; r)$.
  7. For each $i \in F$, the challenger samples $\widetilde{P}_i \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Prog}[\widetilde{K}, \widetilde{\mathsf{hk}}, i])$ and sets $\widetilde{\mathsf{sh}}'_i = (\widetilde{\mathsf{sh}}_i, \widetilde{\mathsf{hk}}, \widetilde{P}_i)$.
  8. The challenger sends $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in F}$ and $\left\{\mathsf{sh}'_i\right\}_{i \in S}$ to $\mathcal{A}$.
  9. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  As we observed after Construction A.4, every faker $i \in F$ derives $P_i(\mathsf{dig}^*, \pi_i) = \mathsf{PPRF.Eval}(K, \mathsf{dig}^*)$. Hence, this is exactly the privacy game of Definition 2.3 instantiated with Construction A.4. Note that the challenger does not need the obfuscated programs P_i of the fakers $i \in F$, since $\mathcal{A}$ only receives their fake shares.

- Hyb^b_k for $k \in [0, t]$: same as Real^b, except that the hash key is binding on s_k, and the programs of the first k snitches are obfuscated with a key that is punctured on the target digest.

  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$, a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, s_k)$, and a master key $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$.
  3. The challenger computes the target digest $\mathsf{dig}^* = \mathsf{SSB.Hash}(\mathsf{hk}, x_F)$ and a punctured key $K^{(\mathsf{dig}^*)} \leftarrow \mathsf{PPRF.Puncture}(K, \mathsf{dig}^*)$.
  4. For each $i \in S$, the challenger samples P_i as follows and sets $\mathsf{sh}'_i = (\mathsf{sh}_i, \mathsf{hk}, P_i)$:

  $$P_i \leftarrow \begin{cases} i\mathcal{O}(1^\lambda, \mathsf{Prog}[K, \mathsf{hk}, i]) & i \notin \{s_1, \ldots, s_k\} \\ i\mathcal{O}(1^\lambda, \mathsf{Prog}[K^{(\mathsf{dig}^*)}, \mathsf{hk}, i]) & i \in \{s_1, \ldots, s_k\} \end{cases}$$

  Note that the challenger still uses the master key K to compute y in step 5.

- $\widetilde{\mathsf{Hyb}}^b_k$ for $k \in [0, t]$: same as Hyb^b_k, except that y is sampled uniformly at random.

  5. The challenger samples $y \xleftarrow{\mathrm{R}} \{0, 1\}^{3\lambda}$ and parses $y = \rho_{\mathsf{hk}} \| \rho_K \| r$.

Clearly, Hyb^b_0 is identical to Real^b, since $s_0 = 1$ and the punctured key is not used. We denote $\mathsf{Hyb}^b := \widetilde{\mathsf{Hyb}}^b_0$, which is identical to Real^b except that $y = \rho_{\mathsf{hk}} \| \rho_K \| r$ is uniformly random.

Lemma A.8. *Assume that Π_{SSB} satisfies index hiding and somewhere statistical binding, Π_{PPRF} is functionality preserving, and $i\mathcal{O}$ is secure. Then for any efficient adversary $\mathcal{A}$, any $b \in \{0,1\}$, and any $k \in [t]$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}^b_{k-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_k(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. We define an intermediate hybrid $\mathsf{Hyb}^b_{k-1,1}$ as follows: The same as Hyb^b_{k-1}, except that the hash key is binding on s_k.

2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$, a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, s_k)$, and a master key $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$.

Hyb^b_{k-1} is indistinguishable from $\mathsf{Hyb}^b_{k-1,1}$. The only difference between the two hybrids is the binding index of hk. We construct an adversary $\mathcal{B}$ for the index hiding game of Π_{SSB}: the adversary $\mathcal{B}$ runs $\mathcal{A}$ to get F (and hence S), sends (n, s_{k-1}, s_k) to the Π_{SSB} challenger and receives hk. It then simulates everything for $\mathcal{A}$ exactly as in Hyb^b_{k-1}, using the hash key hk. This is possible since the challenger in both hybrids never uses the binding index other than for sampling hk. Eventually, algorithm $\mathcal{B}$ outputs whatever bit b' that $\mathcal{A}$ outputs. If hk is binding on s_{k-1}, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b_{k-1}(\mathcal{A})$, and if hk is binding on s_k, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b_{k-1,1}(\mathcal{A})$. The claim now follows by the index hiding of Π_{SSB}.

$\mathsf{Hyb}^b_{k-1,1}$ is indistinguishable from Hyb^b_k. The only difference between the two hybrids is that the program P_{s_k} is an obfuscation of $C_0 = \mathsf{Prog}[K, \mathsf{hk}, s_k]$ in $\mathsf{Hyb}^b_{k-1,1}$ and an obfuscation of $C_1 = \mathsf{Prog}[K^{(\mathsf{dig}^*)}, \mathsf{hk}, s_k]$ in Hyb^b_k. We construct an adversary $\mathcal{B}$ for the security game of $i\mathcal{O}$: the adversary $\mathcal{B}$ samples all the values as in $\mathsf{Hyb}^b_{k-1,1}$, sends (C_0, C_1) to the $i\mathcal{O}$ challenger, and uses the obfuscated program it receives as P_{s_k}. It then simulates everything else exactly as in both hybrids (the master key K is known to $\mathcal{B}$), and outputs whatever bit b' that $\mathcal{A}$ outputs. For $\mathcal{B}$ to be a valid adversary for the security game of $i\mathcal{O}$, the circuits C_0 and C_1 have to be functionally equivalent (they have the same size by padding). We prove next that this is indeed the case whenever hk is binding on s_k, which by the somewhere statistical binding of Π_{SSB} happens with all but negligible probability.

Fix an input (dig, π) to the program. If $\mathsf{SSB.Verify}(\mathsf{hk}, \mathsf{dig}, s_k, 1, \pi) = 0$, then both programs output $\perp$. Otherwise, if $\mathsf{dig} \neq \mathsf{dig}^*$, then by the functionality preserving of Π_{PPRF} we have $\mathsf{PPRF.Eval}(K^{(\mathsf{dig}^*)}, \mathsf{dig}) = \mathsf{PPRF.Eval}(K, \mathsf{dig})$, so both programs output the same value. It remains to show that $\mathsf{SSB.Verify}(\mathsf{hk}, \mathsf{dig}^*, s_k, 1, \pi) = 0$ for every π. Since $s_k \in S$, we have $s_k \notin F$, so the s_k-th bit of x_F is 0. By the correctness of Π_{SSB}, the honest opening $\pi_0 = \mathsf{SSB.Open}(\mathsf{hk}, x_F, s_k)$ satisfies $\mathsf{SSB.Verify}(\mathsf{hk}, \mathsf{dig}^*, s_k, 0, \pi_0) = 1$. Since hk is binding on s_k, there does not exist π such that $\mathsf{SSB.Verify}(\mathsf{hk}, \mathsf{dig}^*, s_k, 1, \pi) = 1$. Thus, the circuits C_0 and C_1 are functionally equivalent. The claim now follows by the security of $i\mathcal{O}$. □

Lemma A.9. *Assume that Π_{SSB} satisfies index hiding and somewhere statistical binding, Π_{PPRF} is functionality preserving, and $i\mathcal{O}$ is secure. Then for any efficient adversary $\mathcal{A}$, any $b \in \{0,1\}$, and any $k \in [t]$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\widetilde{\mathsf{Hyb}}^b_{k-1}(\mathcal{A}) = 1] - \Pr[\widetilde{\mathsf{Hyb}}^b_k(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Lemma A.9 follows by an analogous argument to Lemma A.8, since that argument does not depend on how y is computed.

Lemma A.10. *Assume that Π_{PPRF} satisfies punctured pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $b \in \{0,1\}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}^b_t(\mathcal{A}) = 1] - \Pr[\widetilde{\mathsf{Hyb}}^b_t(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. In Hyb^b_t, all the programs $\{P_i\}_{i \in S}$ given to $\mathcal{A}$ are obfuscated with the punctured key $K^{(\mathsf{dig}^*)}$, and the master key K is only used to compute $y = \mathsf{PPRF.Eval}(K, \mathsf{dig}^*)$. We construct an adversary $\mathcal{B}$ for the punctured pseudorandomness game of Π_{PPRF}: the adversary $\mathcal{B}$ runs $\mathcal{A}$ to get F, samples $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, s_t)$, computes

$\mathsf{dig}^* = \mathsf{SSB.Hash}(\mathsf{hk}, x_F)$, and sends $(1^{\ell_{\mathsf{dig}}}, 1^{3\lambda}, \mathsf{dig}^*)$ to the Π_{PPRF} challenger. It receives a punctured key $K^{(\mathsf{dig}^*)}$ and a challenge y, and uses them to simulate everything for $\mathcal{A}$ exactly as in Hyb_t^b. Eventually, algorithm $\mathcal{B}$ outputs whatever bit b' that $\mathcal{A}$ outputs. If $y = \mathsf{PPRF.Eval}(K, \mathsf{dig}^*)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_t^b(\mathcal{A})$, and if y is uniformly random, then $\mathcal{B}$ perfectly simulates $\widetilde{\mathsf{Hyb}}_t^b(\mathcal{A})$. The claim now follows by the punctured pseudorandomness of Π_{PPRF}. □

Lemma A.11. *Assume that Π_{CSS} is secure. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Since Share, SSB.Setup, and PPRF.KeyGen use exactly λ random bits each, and $\rho_{\mathsf{hk}} \| \rho_K \| r$ is uniform in Hyb^b, the hash key $\widetilde{\mathsf{hk}}$, the master key $\widetilde{K}$, and the shares $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n)$ are distributed as fresh samples of $\mathsf{SSB.Setup}(1^\lambda, n, 1)$, $\mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$, and $\mathsf{Share}(1^\lambda, m^{\mathsf{fake}})$, independently of b, of $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$, of hk, and of K. Therefore, the only part of the view of $\mathcal{A}$ that depends on b is $\{\mathsf{sh}_i\}_{i \in S}$. We construct an adversary $\mathcal{B}$ for the security game of Π_{CSS}: the adversary $\mathcal{B}$ runs $\mathcal{A}$ to get $(F, m_0, m_1, m^{\mathsf{fake}})$, sends (S, m_0, m_1) to the Π_{CSS} challenger, and receives $\{\mathsf{sh}_i\}_{i \in S}$. It then samples hk, K, and $\{P_i\}_{i \in S}$ exactly as in Real^b, and samples the fake shares $\left\{\widetilde{\mathsf{sh}}_i'\right\}_{i \in F}$ from uniform randomness exactly as in Hyb^b. Eventually, algorithm $\mathcal{B}$ outputs whatever bit b' that $\mathcal{A}$ outputs. Since $S \notin \Gamma$, algorithm $\mathcal{B}$ is a valid adversary for the security game of Π_{CSS}. If the Π_{CSS} challenger shares m_b, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b(\mathcal{A})$. The claim now follows by the security of Π_{CSS}. □

Proof of Theorem A.7. Let $\mathcal{A}$ be an efficient adversary for the privacy game against Construction A.4, and fix $b \in \{0, 1\}$. Recall that Real^b is identical to Hyb_0^b. By Lemma A.8, Lemma A.10, and Lemma A.9, we have the following sequence of hybrids

$$\mathsf{Real}^b = \mathsf{Hyb}_0^b \approx \mathsf{Hyb}_1^b \approx \cdots \approx \mathsf{Hyb}_t^b \approx \widetilde{\mathsf{Hyb}}_t^b \approx \widetilde{\mathsf{Hyb}}_{t-1}^b \approx \cdots \approx \widetilde{\mathsf{Hyb}}_0^b = \mathsf{Hyb}^b,$$

where each pair of consecutive hybrids is computationally indistinguishable. Since $t \le n$, this is a polynomial number of hybrids, and therefore by the triangle inequality there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Combined with Lemma A.11 and the triangle inequality, we get that

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

This concludes the proof of Theorem A.7.

Theorem A.12 (Fake-Real Indistinguishability)**.** *Assume that Π_{SSB} satisfies index hiding and somewhere statistical binding (as per Definition A.3), Π_{PPRF} satisfies punctured pseudorandomness (as per Definition A.2), and $i\mathcal{O}$ is a secure indistinguishability obfuscator (as per Definition A.1). Then Construction A.4 satisfies fake-real indistinguishability (as per Definition 2.3).*

Let $\mathcal{A}$ be an adversary for the fake-real indistinguishability game. Throughout the proof, for a bit $b \in \{0, 1\}$, we denote by $P_b = \{s_1, \ldots, s_t\}$ the set of parties that hold real shares, where $s_1 < \cdots < s_t$, and we set $s_0 := 1$. We define the following sequence of hybrids starting with the real fake-real indistinguishability game Real^b.

- Real^b: This is the real fake-real indistinguishability game with parameter b.
  1. On input 1^λ, the adversary $\mathcal{A}$ sends a partition (P_0, P_1) of $[n]$ and secrets $m_0, m_1 \in \mathcal{M}$.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$, a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, 1)$, and a master key $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$.
  3. The challenger computes the target digest $\mathsf{dig}^* = \mathsf{SSB.Hash}(\mathsf{hk}, x_{P_{1-b}})$.
  4. For each $i \in P_b$, the challenger samples $P_i \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Prog}[K, \mathsf{hk}, i])$ and sets $\widetilde{\mathsf{sh}}_i' = (\mathsf{sh}_i, \mathsf{hk}, P_i)$.

5. The challenger computes $y = \mathsf{PPRF.Eval}(K, \mathsf{dig}^*)$ and parses $y = \rho_{\mathsf{hk}} \| \rho_K \| r$.
6. The challenger computes $\widetilde{\mathsf{hk}} = \mathsf{SSB.Setup}(1^\lambda, n, 1; \rho_{\mathsf{hk}})$, $\widetilde{K} = \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda}; \rho_K)$ and $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m_{1-b}; r)$.
7. For each $i \in P_{1-b}$, the challenger samples $\widetilde{P}_i \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Prog}[\widetilde{K}, \widetilde{\mathsf{hk}}, i])$ and sets $\widetilde{\mathsf{sh}}'_i = (\widetilde{\mathsf{sh}}_i, \widetilde{\mathsf{hk}}, \widetilde{P}_i)$.
8. The challenger sends $\left\{\widetilde{\mathsf{sh}}'_i\right\}_{i \in [n]}$ to $\mathcal{A}$.
9. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

As we observed after Construction A.4, every faker $i \in P_{1-b}$ derives $P_i(\mathsf{dig}^*, \pi_i) = \mathsf{PPRF.Eval}(K, \mathsf{dig}^*)$. Hence, this is exactly the fake-real indistinguishability game of Definition 2.3 instantiated with Construction A.4.

- Hyb^b_k for $k \in [0,t]$: same as Real^b, except that the hash key is binding on s_k, and the programs of the first k parties in P_b are obfuscated with a key that is punctured on the target digest.

  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, m_b)$, a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, s_k)$, and a master key $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathsf{dig}}}, 1^{3\lambda})$.
  3. The challenger computes the target digest $\mathsf{dig}^* = \mathsf{SSB.Hash}(\mathsf{hk}, x_{P_{1-b}})$ and a punctured key $K^{(\mathsf{dig}^*)} \leftarrow \mathsf{PPRF.Puncture}(K, \mathsf{dig}^*)$.
  4. For each $i \in P_b$, the challenger samples P_i as follows and sets $\widetilde{\mathsf{sh}}'_i = (\mathsf{sh}_i, \mathsf{hk}, P_i)$:

$$P_i \leftarrow \begin{cases} i\mathcal{O}(1^\lambda, \mathsf{Prog}[K, \mathsf{hk}, i]) & i \notin \{s_1, \ldots, s_k\} \\ i\mathcal{O}(1^\lambda, \mathsf{Prog}[K^{(\mathsf{dig}^*)}, \mathsf{hk}, i]) & i \in \{s_1, \ldots, s_k\} \end{cases}$$

  Note that the challenger still uses the master key K to compute y in step 5.

- $\widetilde{\mathsf{Hyb}}^b_k$ for $k \in [0,t]$: same as Hyb^b_k, except that y is sampled uniformly at random.

  5. The challenger samples $y \xleftarrow{\mathrm{R}} \{0,1\}^{3\lambda}$ and parses $y = \rho_{\mathsf{hk}} \| \rho_K \| r$.

Clearly, Hyb^b_0 is identical to Real^b, since $s_0 = 1$ and the punctured key is not used. We denote $\mathsf{Hyb}^b := \widetilde{\mathsf{Hyb}}^b_0$, which is identical to Real^b except that $y = \rho_{\mathsf{hk}} \| \rho_K \| r$ is uniformly random.

Lemma A.13. *Assume that Π_{SSB} satisfies index hiding and somewhere statistical binding, Π_{PPRF} is functionality preserving, and $i\mathcal{O}$ is secure. Then for any efficient adversary $\mathcal{A}$, any $b \in \{0,1\}$, and any $k \in [t]$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}^b_{k-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_k(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Lemma A.13 follows by an analogous argument to Lemma A.8, where P_b plays the role of the snitches and P_{1-b} plays the role of the fakers. Indeed, that argument only relies on the fact that the s_k-th bit of the hashed string is 0. This holds here since $s_k \in P_b$ and (P_0, P_1) is a partition, so $s_k \notin P_{1-b}$. In particular, the argument does not rely on P_b being unauthorized.

Lemma A.14. *Assume that Π_{PPRF} satisfies punctured pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $b \in \{0,1\}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}^b_t(\mathcal{A}) = 1] - \Pr[\widetilde{\mathsf{Hyb}}^b_t(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Lemma A.14 follows by an analogous argument to Lemma A.10.

Lemma A.15. *Assume that Π_{SSB} satisfies index hiding and somewhere statistical binding, Π_{PPRF} is functionality preserving, and $i\mathcal{O}$ is secure. Then for any efficient adversary $\mathcal{A}$, any $b \in \{0,1\}$, and any $k \in [t]$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\widetilde{\mathsf{Hyb}}^b_{k-1}(\mathcal{A}) = 1] - \Pr[\widetilde{\mathsf{Hyb}}^b_k(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Lemma A.15 follows by an analogous argument to Lemma A.13, since that argument does not depend on how y is computed.

Lemma A.16. *For any (unbounded) adversary $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1].$$

Proof. Since Share, SSB.Setup, and PPRF.KeyGen use exactly λ random bits each, and $\rho_{\mathsf{hk}} \| \rho_K \| r$ is uniform in Hyb^b, the hash key $\widetilde{\mathsf{hk}} = \mathsf{SSB.Setup}(1^\lambda, n, 1; \rho_{\mathsf{hk}})$, the master key $\widetilde{K} = \mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathrm{dig}}}, 1^{3\lambda}; \rho_K)$, and the shares $(\widetilde{\mathsf{sh}}_1, \ldots, \widetilde{\mathsf{sh}}_n) = \mathsf{Share}(1^\lambda, m_{1-b}; r)$ are distributed as fresh samples of $\mathsf{SSB.Setup}(1^\lambda, n, 1)$, $\mathsf{PPRF.KeyGen}(1^\lambda, 1^{\ell_{\mathrm{dig}}}, 1^{3\lambda})$, and $\mathsf{Share}(1^\lambda, m_{1-b})$, independently of $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$, of hk, and of K. Therefore, the view of $\mathcal{A}$ in Hyb^b is distributed as follows:

- For each $i \in P_b$, the share $\widetilde{\mathsf{sh}}'_i$ consists of the i-th share of a fresh sharing of m_b, a fresh hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, 1)$, and an obfuscation of $\mathsf{Prog}[K, \mathsf{hk}, i]$ for a fresh master key K.
- For each $i \in P_{1-b}$, the share $\widetilde{\mathsf{sh}}'_i$ consists of the i-th share of a fresh sharing of m_{1-b}, a fresh hash key $\widetilde{\mathsf{hk}} \leftarrow \mathsf{SSB.Setup}(1^\lambda, n, 1)$, and an obfuscation of $\mathsf{Prog}[\widetilde{K}, \widetilde{\mathsf{hk}}, i]$ for a fresh master key $\widetilde{K}$.

Moreover, the two sharings, the two hash keys, and the two master keys are mutually independent. In other words, for every $\beta \in \{0, 1\}$, the shares of P_β consist of a fresh sharing of m_β together with a fresh hash key and obfuscated programs of a fresh master key, independently of the shares of $P_{1-\beta}$. This distribution does not depend on b, and the lemma follows. □

Proof of Theorem A.12. Let $\mathcal{A}$ be an efficient adversary for the fake-real indistinguishability game against Construction A.4. Fix $b \in \{0, 1\}$. Recall that Real^b is identical to Hyb_0^b. By Lemma A.13, Lemma A.14, and Lemma A.15, we have the following sequence of hybrids

$$\mathsf{Real}^b = \mathsf{Hyb}_0^b \approx \mathsf{Hyb}_1^b \approx \cdots \approx \mathsf{Hyb}_t^b \approx \widetilde{\mathsf{Hyb}}_t^b \approx \widetilde{\mathsf{Hyb}}_{t-1}^b \approx \cdots \approx \widetilde{\mathsf{Hyb}}_0^b = \mathsf{Hyb}^b,$$

where each pair of consecutive hybrids is computationally indistinguishable. Since $t \le n$, this is a polynomial number of hybrids, and therefore by the triangle inequality there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Combined with Lemma A.16 and the triangle inequality, we get that

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

This concludes the proof of Theorem A.12.